Automated Knowledge Base Quality Assessment and Validation based on Evolution Analysis

ScuDo

Scuola di Dottorato ~ Doctoral School

WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation

Doctoral Program in Energy Engineering (30^{th}cycle)

Automated Knowledge Base Quality Assessment and Validation based on Evolution Analysis

By

Mohammad Rifat Ahmmad Rashid

******

Supervisor(s):

Prof. Marco Torchiano

Doctoral Examination Committee:

Prof. Mario Piattini, Referee, Universidad de Castilla-La Mancha, Spain

Dr. Anastasia Dimou, Referee, RUG - Universiteit Gent, Belgium

Dr. Giuseppe Rizzo, Istituto Superiore Mario Boella, Italy

Prof. Fulvio Corno, Politecnico di Torino, Italy

Prof. Maurizio Morisio, Politecnico di Torino, Italy

Politecnico di Torino

2018

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Mohammad Rifat Ahmmad Rashid
2018

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

I would like to dedicate this thesis to my loving parents and my sister Farhana Rashid for being my source of motivation and inspiration. I could not have asked for better sister.

To my teachers who guided me in this education process.

Acknowledgements

With this thesis, my Ph.D. journey is reaching the end. I would like to thank all of the people who contributed in some way during this journey. First and foremost, I would like to thank my supervisor Prof. Marco Torchiano for his support and confidence in me. I am grateful for all the support and the valuable discussions and advices, and the chance to deal with situations which I though beyond my capabilities at the time. Then, I would like to thank my advisor, Dr. Giuseppe Rizzo, your open mind, pragmatism, and valuable feedback helped me to grow both intellectually and personally.

I am also grateful to Prof. Maurizio Morisio for his availability and for sharing his experience. I also appreciated the precious help in technical subjects and the hints to deal with the bureaucratic issues received by Luca Ardito.

I would like to express sincere gratitude to all the colleagues of the SoftEng group. First of all, thanks to Cristhian Figueroa, Iacopo Vagliano, and Oscar Rodriguez for the encouragement and suggestions that motivated me a lot in these years. Indeed, I would like to mention all the mates who made the atmosphere in the Lab 1 more pleasant: Erion Çano, Riccardo Coppola, Diego Monti, Francesco Strada, Amirhosein Toosi and Alysson Dos Santos. A very special thanks to Diego Monti for his help during my thesis writing and the bureaucratic procedures.

An outstanding experience was the research visiting at the Universidad Politécnica de Madrid, in Spain. I would like to thank all the colleagues of the Ontology Engineering Group, in particular, Prof Oscar Corcho, who supervised me, Nanadan Mihindukulasooriya for the useful discussions, and for the results reached through our joint effort.

Also, I would like to thank TIM (formerly Telecom Italia) for granting me the scholarship, without which pursuing a PhD would be more difficult. I express my

gratitude to Marco Marengo and JOL Mobile team for cooperative work conducted together.

Most importantly, thank you to my mother, father, and sisters. Thank you for teaching me the real values of life. Specially thanks to my sister Farhana Rashid for your unconditional support. I have never known a day apart from your love, and that is a priceless gift to me. I hope my life and my choices have honored you.

Abstract

In recent years, numerous efforts have been put towards sharing Knowledge Bases (KB) in the Linked Open Data (LOD) cloud. These KBs are being used for various tasks, including performing data analytics or building question answering systems. Such KBs evolve continuously: their data (instances) and schemas can be updated, extended, revised and refactored. However, unlike in more controlled types of knowledge bases, the evolution of KBs exposed in the LOD cloud is usually unrestrained, what may cause data to suffer from a variety of quality issues, both at a semantic level and at a pragmatic level. This situation affects negatively data stakeholders – consumers, curators, etc. –. Data quality is commonly related to the perception of the *fitness for use*, for a certain application or use case. Therefore, ensuring the quality of the data of a knowledge base that evolves is vital. Since data is derived from autonomous, evolving, and increasingly large data providers, it is impractical to do manual data curation, and at the same time, it is very challenging to do a continuous automatic assessment of data quality. Ensuring the quality of a KB is a non-trivial task since they are based on a combination of structured information supported by models, ontologies, and vocabularies, as well as queryable endpoints, links, and mappings. Thus, in this thesis, we explored two main areas in assessing KB quality: (i) quality assessment using KB evolution analysis, and (ii) validation using machine learning models.

The evolution of a KB can be analyzed using fine-grained "change" detection at low-level or using "dynamics" of a dataset at high-level. In this thesis we present a novel knowledge base quality assessment approach using evolution analysis. The proposed approach uses data profiling on consecutive knowledge base releases to compute quality measures that allow detecting quality issues. However, the first step in building the quality assessment approach was to identify the quality characteristics. Using high-level change detection as measurement functions, in this thesis we present four quality characteristics: Persistency, Historical Persistency, Consistency,

and Completeness. Persistency and historical persistency measures concern the degree of changes and lifespan of any entity type. Consistency and completeness measures identify properties with incomplete information and contradictory facts. The approach has been assessed both quantitatively and qualitatively on a series of releases from two knowledge bases, eleven releases of DBpedia and eight releases of 3cixty Nice.

However, high-level changes, being coarse-grained, cannot capture all possible quality issues. In this context, we present a validation strategy whose rationale is twofold. First, using manual validation from qualitative analysis to identify causes of quality issues. Then, use RDF data profiling information to generate integrity constraints. The validation approach relies on the idea of inducing RDF shape by exploiting SHACL constraint components. In particular, this approach will learn, what are the integrity constraints that can be applied to a large KB by instructing a process of statistical analysis, which is followed by a learning model. We illustrate the performance of our validation approach by using five learning models over three sub-tasks, namely minimum cardinality, maximum cardinality and range constraint.

The techniques of quality assessment and validation developed during this work are automatic and can be applied to different knowledge bases independently of the domain. Furthermore, the measures are based on simple statistical operations that make the solution both flexible and scalable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Linked Data approach consists in exposing and connecting data from different sources on the Web by the means of semantic web technologies. Tim Berners-Lee[1] refers to linked open data as a distributed model for the Semantic Web that allows any data provider to publish its data publicly, in a machine readable format, and to meaningfully link them with other information sources over the Web. This is leading to the creation of the Linked Open Data (LOD)[2] cloud consisting of several Knowledge Bases (KBs) that make available billions of RDF[3] triples from different domains such as Geography, Government, Life Sciences, Media, Publication, Social Networking, and User generated data [2].

Such KBs evolve over time: their data instances and schemes can be updated, extended, revised and refactored [3]. However, unlike what happens in more controlled types of knowledge bases, the evolution of KBs exposed in the LOD cloud is usually unrestrained, which may cause data to suffer from a variety of quality issues, both at a semantic level and at a data instance level. By considering the aggregated measure of conformance, the empirical study carried out by Debattista *et al.* [4] shows that datasets published in the LOD cloud have reasonable quality. Nevertheless, it also pointed out that significant issues remain concerning individual quality metrics, such as data provenance and licensing. We can explore certain quality issues by looking at individual metrics, for example quality issues in the data collection or integration processes.

[1] http://www.w3.org/DesignIssues/LinkedData.html
[2] http://lod-cloud.net
[3] https://www.w3.org/RDF

Data quality relates to the perception of the "fitness for use" in a given context [5]. In this thesis, we considered *quality assessment* as the process of statistically assessing the evolving KB resources to identify anomalies that can affect the knowledge base exploitation and any application usage. Furthermore, we use the term *validation* to indicate the principal mean of evaluating the performance of quality assessment procedure. Also, *validation* approaches allow the data stakeholders, such as consumer and curator, to monitor the results thoroughly and provide feedback regarding the ability to identify possible issues.

Ensuring the data quality of a knowledge base that evolves over time is vital. Since data is derived from autonomous, evolving, and increasingly large data providers, it is impractical to perform manual data curation tasks, and at the same time, it is very challenging to perform a continuous automatic assessment of data quality. In this context, KB evolution can be analyzed using fine-grained "change" detection at low-level or using "dynamics" of a dataset at high-level. Fine-grained changes of KB sources are analyzed with regard to their sets of triples, set of entities, or schema signatures [6–8]. For example, fine-grained analysis at the triple level between two snapshots of a KB can detect which triples from the previous snapshot have been preserved in the later snapshot. Moreover, fine-grained analysis can detect which triples have been deleted, or which ones have been added [2]. The dynamic feature of a dataset give insights into how it behaves and evolves over a certain period [7]. Ellefi *et al.* [9] explored the dynamic features considering the use cases presented by Käfer *et al.* [10]. *KB evolution analysis* using dynamic feature help to understand the changes applied to an entire KB or parts of it. It has multiple dimension regarding the dataset update behavior, such as frequency of change, change patterns, change impacts, and causes of change. More specifically, using dynamicity of a dataset, we can capture those changes that happen often; or changes that the curator wants to highlight because they are useful or interesting for a specific domain or application; or changes that indicate an abnormal situation or type of evolution [7, 8]. Table 1.1 summarizes the features of the two types of analysis.

One of the common tasks for data quality assessment is to perform a detailed data analysis with data profiling [11]. Data profiling is usually defined as the process of examining data to collect statistics and provide relevant metadata about the data [12]. Based on data profiling we can thoroughly examine and understand a KB, its structure, and its properties before usage. For example, the DBpedia KB [13] has

Table 1.1 Summary of the two analysis type features.

Analysis level	Detail	Volume	Stakeholder
Low-level	fine-grained	Large	Data end-user
High-level	coarse-grained	Small	Data Curator

been already available for a long time, with various versions that have been released periodically. Along with each release, the DBpedia KB implemented changes at both the instance and schema level. The changes at the schema level involve classes, properties, axioms, and mappings to other ontologies [14]. Instance level changes include resource typing, property values, or identifying links between resources. Based on the data profiling information we can detect changes between various releases. Thus, in this thesis, we investigate the following research areas:

- *KB evolution analysis using data profiling to identify quality issues.*
- *Validation based on integrity constraints using data profiling information and learning models.*

1.1 Problem statement

In this thesis, we argue that evolving KBs suffers from issues such as erroneous instances that get removed or that are added with wrong semantics. Such a phenomenon affects data stakeholders, such as consumers and curators, negatively. Our work explores two main challenges of evolving KBs: *(i)* identification of quality issues due to unrestrained KB evolution, and *(ii)* identification of logical or formal contradiction, which we outline in this section.

1.1.1 Identification of quality issues for evolving KBs

Assessing the quality of an evolving knowledge base is a challenging task, as it often requires to identify correct quality assessment procedures. Also, the quality assessment process is affected by the evolution of data instances, ontologies and vocabularies, and queryable endpoints. Furthermore, the KB evolution can directly

impact the data integration tasks (e.g., synchronization, data linking or fusion), that may lead to incomplete or incorrect results [15]. For example, DBpedia collects data from semi-structured sources which is created in a crowdsourcing effort (i.e., Wikipedia). This extracted data might have quality problems because it is either mapped incorrectly or the source information itself is incorrect.

KB evolution can be explored based on simple changes at low-level and complex changes at high-level [8]. In particular, performing a fine-grained analysis based on low-level changes means substantial data processing challenges. On the contrary, a coarse-grained analysis using high-level changes can help to obtain an approximate indication of the quality a data curator can expect. However, high-level change detection at the instance level, being coarse-grained, cannot capture all possible quality issues.

In general, low-level changes are easy to define and have several interesting properties [8]. Low-level change detection compares the current with the previous dataset version and returns the delta containing the added or deleted entities. For example, two DBpedia versions – 201510 and 201604 – have the property *dbo:areaTotal* in the domain of *dbo:Place* (the prefix dbo: present as an alias for the DBpedia ontology namespace: <http://dbpedia.org/ontology/>). Low-level changes can help to detect added or deleted instances for *dbo:Place* entity type. One of the main requirements for quality assessment would be to identify the completeness of *dbo: Place* entity type with each KB releases. Low-level changes can help only to detect missing entities with each KB release. Such as those entities missing in the 201604 version (e.g. dbr:A_Rúa, dbr:Sandiás,dbr:Coles_Qurense) [4]. Furthermore, these instances are automatically extracted from Wikipedia Infobox keys. We track the Wikipedia page from which DBpedia statements were extracted. These instances are present in the Wikipedia Infobox as Keys but missing in the DBpedia 201604 release. Thus, for a large volume of the dataset, it is a tedious, time-consuming, and error-prone task to generate such quality assessment manually.

The representation of changes at low-level leads to syntactic and semantic deltas [16] from which it is more difficult to get insights to complex changes or changes intended by a human user. On the contrary, high-level changes can capture the changes that indicate an abnormal situation and generates results that are intu-

[4]Here the prefix dbr: present as an alias for the DBpedia resources namespace: <http://dbpedia.org/resource/>

itive enough for the human user. High-level changes from the data can be detected using statistical profiling. For example, total entity count of *dbo:Place* type for two DBpedia versions – 201510 and 201604 – is 1,122,785 and 925,383 where the entity count of 201604 is lower than 201510. This could indicate an imbalance in the data extraction process without fine-grain analysis. However, high-level changes require a fixed set of requirements to understand underlying changes happening in the dataset. For example, assuming that the schema of a KB remains unchanged, a set of low-level changes from data correspond to one high-level change.

1.1.2 Identification of logical or formal contradiction

A data quality analysis using high-level change detection may lead to increasing the number of false positives if the version of a KB is deployed with design issues, such as incorrect mappings. Furthermore, without proper data management, the dataset in an evolving KB may contain consistency issues [17]. Another issue of unrestrained KB evolution is the unavailability of explicit schema information that precisely defines the types of entities and their properties [7]. In general, RDF has proven to be a good model for data integration, and there are several applications using RDF either for data storage or as an interoperability layer [8]. However, unavailability of schema information remains as one of the drawbacks of the RDF data model.

In particular, a knowledge base is defined to be consistent if it does not contain conflicting or contradictory data [18]. When a schema is available with integrity constraints, the data usually goes through a validation process that verifies the compliance against those constraints. Those integrity constraints encapsulate the consistency requirements of data in order to fit for a set of use cases. For example, in a relational database, the integrity constraints are expressed in a data definition language (DDL), and the database management system (DBMS) ensures that any data inserted into the entire database will not lead to any inconsistency. In this context, various research endeavors focus on validation of RDF data to identify any logical or formal contradiction [19].

Validation of RDF data is not done in the same manner as traditional database management systems due to several reasons. One of the key reason is the lack of a language for expressing constraints or having generic models suitable for wider use and not for specific use cases. For example, in DBpedia KB [13] (version 2016-04),

a person should have exactly one value for the "dbo:birthDate" property or the values of the "dbo:height" property should always be a positive number. The instances of the Person class have more than 13,000 associated properties (including dbo, DBpedia ontology properties and dbp, auto-generated properties from Wikipedia infobox keys). Furthermore, ontologies are usually designed for entailment purposes rather than for assessment and their representation often lacks the granular information needed for validating constraints in the data. Taking into account ontology evolution and data quality, in the empirical study presented by Mihindukulasooriya *et al.* [20] explicitly pointed out that changes in the ontology depend on the development process and the community involved in the creation of the knowledge base. They also pointed out the drawbacks of finding practical guidelines and best practices for ontology evolution. This lead to the need for automatic consistency analysis for evolving KBs.

1.2 Research Questions and Contributions

This thesis is based on the conference and journal publications presented in Appendix B, in which I have been an author or a contributor. In this section, we outline the key research questions (RQ) that address the aforementioned problems along with our contributions towards each of them.

1.2.1 KB evolution-based quality assessment

We address the challenges of quality assessment for evolving KB using dynamic features from data profiling. We propose a KB quality assessment approach using quality measures that are computed using KB evolution analysis. Based on the high-level change detection, we aim to analyze quality issues in any knowledge base. The main hypothesis that has guided our investigation is:

Dynamic features from data profiling can help to identify quality issues.

We divide this research goal into two research questions:

RQ1: *How can we identify quality issues with respect to KB evolution?*

In response to this question, we explored the guidelines from two data quality standards, namely ISO/IEC 25024 [1] and W3C DQV [21]. We also explored

the comprehensive survey presented by Zaveri *et al.* [22] on linked open data quality. We propose four quality characteristics based on change detection of a KB over various releases. We use basic statistics (i.e., counts, and diffs) over entities from various KB releases to measure the quality characteristics. More specifically, measurement functions are built using entity count and the amount of changes between pairs of KB releases. Our proposed evaluation-based quality characteristics (RQ1) and approach (RQ2) are presented in a paper entitled "A Quality Assessment Approach for Evolving Knowledge Bases", accepted for publication in the Semantic Web Journal (SWJ) [2]. Furthermore, details on quality assessment and validation approach for specific use cases are presented in a paper entitled "Automated Quality Assessment and Validation for Evolving Knowledge Bases", under review for publication in the *Journal of Web Semantics* [23]. We report the conceptualization of KB evolution analysis and details of evolution-based quality characteristics in Chapter 4.

RQ2: *Which quality assessment approach can be defined on top of the the evolution-based quality characteristics?*

To address this question, we present a quality assessment approach for analyzing quality issues using dynamic features over different KB releases. The main idea behind our quality assessment approach based on the fact that it is not sufficient to identify what is changed, but how to interact with changes and evolving attributes of a KB. Using data profiling, we explored dynamic features on the class level and the property level. We thus evaluate a KB using proposed four evolution-based quality characteristics. For validation, we further explored data profiling information to generate SHACL based integrity constraints for consistency checks. In particular, we learn what are the integrity constraints that can be applicable in a large KB by instructing a process of statistical analysis that is followed by a learning model.

We created KBQ, a tool that automates the detection and report generation of quality issues for evolving knowledge bases. Evolution-based quality assessment approach presented in [2], and details about the KBQ-tool has been published in a paper entitled "KBQ - A Tool for Knowledge Base Quality Assessment Using Evolution Analysis" in the K-CAP workshops of Machine Reading [24]. We provide the detailed explanation of the evolution-based quality assessment approach in Chapter 6.

1.2.2 Validation approaches leveraging on quality characteristics and integrity constraints.

To validate the quality assessment results, in Chapter 6, we present an experimental analysis that is based on a quantitative, qualitative and constraints-based validation approach. We propose RDF shape induction approach to address the challenges of logical or formal contradiction identification in an evolving KB. Regarding the constraints-based validation approach, we derived the following hypothesis:

Learning models can be used for validation using data profiling information as predictive features.

We present this research goal into the following research question:

RQ3: *Which approaches can be used to validate a KB evolution based quality assessment approach?*

In order to validate the proposed quality characteristics, we performed both quantitative and qualitative analysis. In particular, quantitative analysis is performed using quality characteristics which are introduced in Chapter 4. Further, we performed qualitative analysis using manual validation to compute precision by examining the results from the quantitative analysis. In Chapter 7, we report about the experimentation of this approach on two KBs: DBpedia [13] (encyclopedic data) and 3cixty [25] (contextual tourist and cultural data). We illustrate the quantitative and qualitative analysis results based on the work presented in [2].

Furthermore, we have introduced an RDF validation approach that relies on SHACL based integrity constraints using Shape induction in Chapter 5. Based on the results from qualitative analysis, we have performed the RDF validation using the dataset from the 3cixty KB and the DBpedia KB. In Chapter 7, we reported the learning model performance of RDF Shape validation approach based on the work presented in [26].

Based on the data profiling information, we present a set of features taking into account inconsistency issues. We applied these features in the learning model to create RDF Shapes. The core concepts regarding RDF Shape induction for creating integrity constraints have been published in the paper entitled "RDF Shape Induction using Knowledge Base Profilings" in *Proceedings of SAC2018: Symposium on*

Applied Computing 2018 [26]. The details of the RDF Shape induction process using SHACL representation of cardinality, range, and string constraints presented in Chapter 5. We chose cardinality constraints to identify any logical contradiction due to wrong mappings in a evolving KB. Similarly, using range constraints values, we explored if there are any type mismatch due to inconsistent KB updates. In Figure 1.1, we present the process flow of our validation approaches that is divided into three main steps:

(i) KB evolution analysis: The target entity types for validation tasks are selected based on the results from the evolution-based quality characteristics.

(ii) Quality Assessment and Validation approach: Using the information from KB evolution analysis and statistical profiler, we perform quality assessment and RDF Shape induction for validation task.

(iii) Evaluation: We perform evaluation using quantitative, qualitative, and constraints based validation. Furthermore, we apply various learning models to assess the constraints based feature dataset for a given class and formulate a RDF Shape.

Fig. 1.1 The process flow of our data-driven procedure for quality assessment and validation.

1.3 Thesis structure

As illustrated in Figure 1.2, this thesis is divided into nine chapters, as follows: Chapter 2 presents background and motivational examples that demonstrate important aspects of our quality assessment and validation approach; Chapter 3 provides an overview of the state-of-the-art which is part of this thesis, focusing on Linked

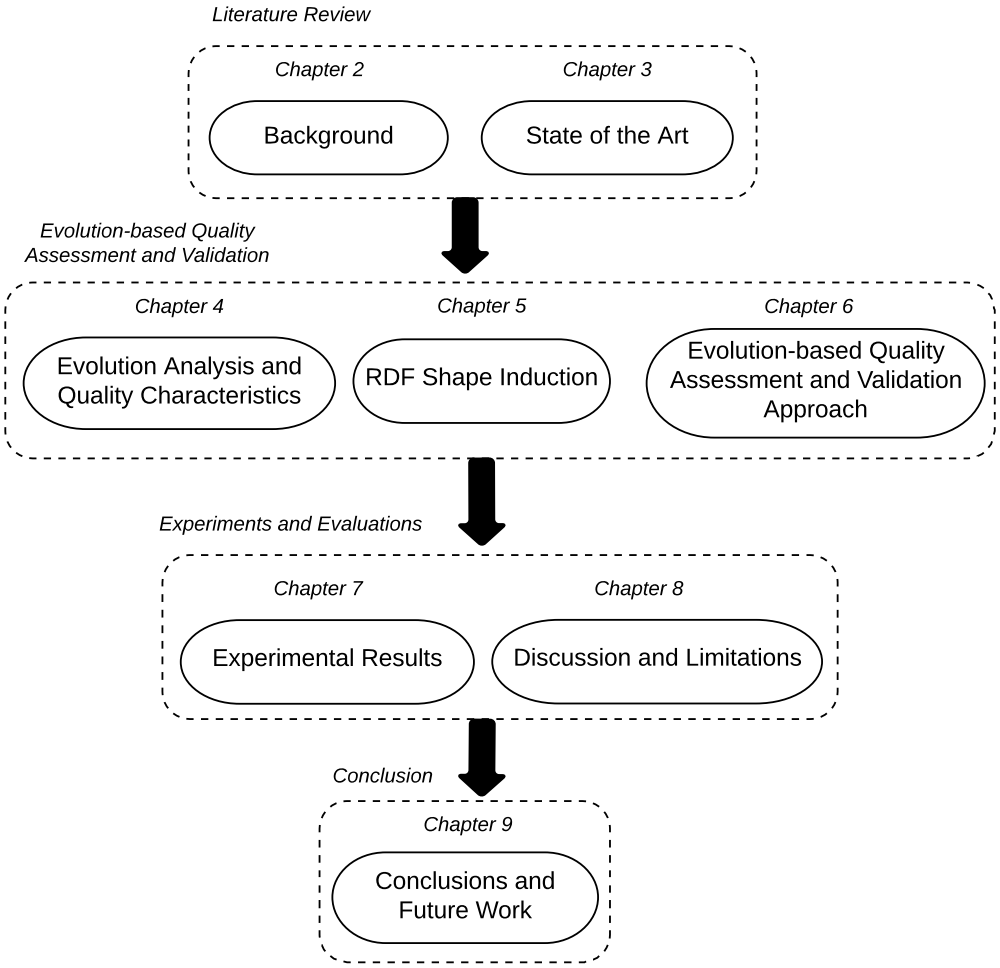


Fig. 1.2 Overview of the thesis structure.

data dynamics, Knowledge Base quality assessment and Knowledge Base validation approaches. Chapter 4 contains the definition of the proposed evolution-based quality characteristics and measurement functions. Chapter 5 describes the process of RDF Shape Induction using SHACL based constraints. Chapter 6 outlines our data driven quality assessment and validation approach; Chapter 7 presents an experimental analysis based on two KBs, namely DBpedia and 3cixty. Furthermore, we considered both English and Spanish versions of the DBpedia KB. Chapter 8 discusses the initial hypothesis, the research questions and insights gathered from the experimentation. Finally, Chapter 9 summarizes the main findings and outlines the future research activities.

Chapter 2

Background

In the following chapter we give an overview of the technical foundations and the background for the reader to understand the work presented in this thesis. The key conceptual definitions are mainly based on [19][1].

In section 2.1 we introduce the reader with the basic concepts of Resource Description Framework (RDF). Section 2.2 gives an overview of RDF related core technologies. An overview of Shapes Constraint Language (SHACL) presented in Section 2.3. Furthermore, Section 2.4 presents the definition of RDF Knowledge base and their evolution. Section 2.4.1 presents an description of the two main KBs namely, 3cxity and DBpedia used in our experimental analysis. A brief overview of data quality standards are presented in Section 2.5. Moreover, Section 2.6 describes the quality issues present in a KB due to evolution. The details of the quality issues are based on the work presented in [2]. Section 2.7 outlines the gold standard creation strategies. Finally, Section 2.8 present an overview of machine learning approaches used in this thesis.

[1] Standard components of the RDF framework and definitions for them are taken from this book as they follow the defined standards and are widely used. Examples for each component are provided by the author.

Table 2.1 Common Prefixes used over the thesis.

Prefix	Namespace
dbo	DBpedia ontology: <http://dbpedia.org/ontology/>
dbr	DBpedia resources:<http://dbpedia.org/resource/>
foaf	FOAF Vocabulary Specification: <http://xmlns.com/foaf/0.1/>
wikipedia-en	English Wikipedia: <https://en.wikipedia.org/wiki/>
lode	3cixty event type: <http://linkedevents.org/ontology>
dul	3cixty place type: <http://www.ontologydesignpatterns.org/ont/dul/DUL.owl>

2.1 Resource Description Framework (RDF)

Resource Description Framework (RDF)[2] is a graph-based data model. RDF datasets produced by different data sources and can be integrated with other data using semantic web technologies. Data integration using RDF is faster and more robust than traditional solutions [19]. The RDF data model is based on the concept of triples. Each triple consists of a subject, a predicate, and an object. RDF triples are usually depicted as a directed arc connecting two nodes (subject and object) by an edge (predicate). An RDF triple asserted means that some relationship, indicated by the predicate, holds between the resources denoted by the subject and object. This is known as an RDF statement. The predicate is an IRI [19] that denotes a property.

Most RDF formats include some mechanism called prefix declaration which enables to simplify writing long IRIs declaring prefix labels. A prefix label associates an alias with an IRI and enables the definition of prefixed names. A prefixed name contains a prefix label and a local part separated by ":" and represents the IRI formed by concatenating the IRI associated with the prefix label and the local part. For example, if ex is declared as a prefix label to represent <http://example.org/>, then ex:rifat is a prefixed name that represents <http://example.org/rifat>. Table 2.1 reports the common prefixes used over the thesis.

An RDF statement can be thought of as a binary relation identified by the property between the subject and object. RDF support other serialization formats like Turtle [3], Trig[4] and JSON-LD[5]. Turtle allows an RDF graph to be completely written in a

[2] https://www.w3.org/RDF
[3] https://www.w3.org/TR/turtle/
[4] https://www.w3.org/TR/trig/
[5] https://www.w3.org/TR/json-ld/

compact and natural text form, with abbreviations for common usage patterns and data types. The code in 2.1 represents an RDF graph in Turtle. The first three lines are prefix declarations, and the rest represents a sequence of RDF triples separated by dots. There are three kinds of nodes: IRIs, literals, and blank nodes.

Listing 2.1 Simple RDF file in Turtle

```
@prefix ex: <http://example.org/>
@prefix schema: <http://schema.org/>
@prefix dbr: <http://dbpedia.org/resource/>

ex:erion schema:knows ex:rifat .
ex:rifat schema:knows ex:diego .
ex:rifat schema:name "Rashid" .
ex:rifat schema:birthDate "1987 -08 -01"^^ xsd:date .
ex:rifat schema:birthPlace dbr:Khulna .
ex:erion schema:knows ex:diego .
ex:erion schema:knows ex:rifat .
ex:erion schema:birthPlace dbr:Tirana .
```

IRI[6] An IRI (Internationalized Resource Identifier) refers to a resource (the referent). A resource can be any thing. IRIs can appear as subjects, predicates and objects. In Turtle, IRIs are enclosed by < and >. For example, an IRI can be <http://example.org/rifat>.

Literals A literal denotes resources which have an associated value, for example, an integer or string value. Literals can only appear as objects in triples.

Blank Node A blank node refers to local identifiers which do not identify any specific resources. It can be used as subjects or objects of triples. They specify that something with the given relationship exists, without explicitly naming it.

2.2 RDF Related Core Technologies

RDF was designed to used as a central piece for knowledge representation in the Web. The goal is that agents can automatically infer new knowledge in the form of

[6] https://www.rfc-editor.org/info/rfc3987

new RDF statements from existing RDF graphs. To that end, several technologies were proposed to increase RDF expressiveness. In this context, various technologies were proposed to support RDF representation. There are three concepts that are commonly utilized in RDF: SPARQL, RDF Schema and OWL.

SPARQL (SPARQL Protocol and RDF Query Language)[7] is an RDF query language which is able to retrieve and manipulate data stored in RDF. SPARQL is based on the notion of Basic Graph Patterns which are sets of triple patterns. A triple pattern is an extension of an RDF triple where some of the elements can be variables which are denoted by a question mark. For example, the following SPARQL query in 2.2 retrieves the nodes ?s whose birth place is dbr:Khulna and the nodes ?o that are known by them.

Listing 2.2 Simple SPARQL Query.

```
@prefix schema: <http://schema.org/>
@prefix dbr: <http://dbpedia.org/resource/>

SELECT ?s ?o WHERE {
?s schema:birthPlace dbr:Khulna .
?s schema:knows ?o
}
```

RDF Schema[8] provides a data-modelling vocabulary for RDF data. It is a semantic extension of RDF which provides mechanisms to describe groups of resources and relationships between them. It defines a set of common classes and properties. The main classes defined in RDFS are:

- *rdfs:Resource:* the class of everything;
- *rdfs:Class:* the class of all classes;
- *rdfs:Literal:* the class of all literal values;
- *rdfs:Datatype:* the class of all datatypes;
- *rdf:Property:*the class of all properties.

[7] https://www.w3.org/TR/rdf-sparql-query/
[8] https://www.w3.org/TR/rdf-schema/

OWL (Web Ontology Language)[9] defines a vocabulary for expressing ontologies based on description logics. OWL has several syntaxes: an RDF-based syntax, functional-style Syntax, manchester syntax, and a formally defined meaning. An ontology can be defined as a vocabulary of terms, usually about a specific domain and shared by a community of users. Ontologies specify the definitions of terms by describing their relationships with other terms in the ontology. The main concepts in OWL are as follows.

- *Classes* which represent sets of individuals. Classes can be subclasses of other classes, with two special classes: *owl:Thing* that represents the set of all individuals and *owl:Nothing* that represents the empty set.
- *Individuals* which are elements in the domain. Individuals can be members of an OWL class.
- *Properties* which represent relationships. Properties are classified as datatype properties, object properties and annotation properties. Datatype properties relate an individual with a data value such as a string or integer. Object properties relate an individual with another individual, and Annotation properties encode information about the ontology itself (such as the author or the creation date of an ontology).
- *Constructors* which allow to define complex concepts from other concepts using expressions.

2.3 Shapes Constraint Language (SHACL)

Shapes Constraint Language (SHACL) has been developed by the W3C RDF Data Shapes Working Group[10], which was initiated in 2014 with the goal to introduce a language for imposing structural constraints on RDF graphs. SHACL structures are motivated by other constraints based languages such as SPIN[11], some parts from OSLC resource shapes[12] and Shape Expression(ShEx)[13]. More specifically, SHACL is inspired by SPIN. The SPIN modeling vocabulary based on RDF properties and

[9] https://www.w3.org/OWL/
[10] https://www.w3.org/2014/data-shapes
[11] http://spinrdf.org/spin.html
[12] http://open-services.net/resources/
[13] https://www.w3.org/2001/sw/wiki/ShEx

classes using SPARQL to specify rules and logical constraints. In the beginning, the main motivation of this initiative is to combine all the constraint languages into SHACL. However, due to core difference of representation, ShEx is not completely converged with SHACL.

SHACL is divided into two parts: *(i)* SHACL Core describes a core RDF vocabulary to define common shapes and constraints; and *(ii)* extension mechanism regarding SPARQL and has been called: SHACL-SPARQL. In this thesis, we explored the SHACL Core for RDF validation process. More specifically, we looked at SHACL Shape for a specific class to identify constraints components. In SHACL, a Shape is defined as the collection of targets and constraints components. Furthermore, targets specify which nodes in the data graph must conform to a shape and constraints components determine how to validate a node. In this context, *Shapes graph* represent an RDF graph that contains shapes, and *Data graph* represents an RDF graph that contains data to be validated. Furthermore, SHACL defines two types of Shapes: *(i) Node shapes* presents the constraints information about a given focus node; and *(ii) Property shapes* present constraints about a property and values of a path for a node.

For example, let us consider a Person Shape which is based on only name and email. Person name has an ”xsd:string” type and email addresses are presented with IRI. In this account, we present a Person SHACL Shape graph in 2.3.

Listing 2.3 A simple example of Person SHACL Shape Graph

```
@prefix: <http://example.org/>
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@schema schema: <http://schema.org/> .

:PersonShape a sh:NodeShape;
    sh:targetNode :rifat, :erion, :diago;
    st:targetClass :Person;
    sh:property [
        sh:path schema:name;
        sh:minCount 1;
        sh:maxCount 1:
        sh:datatype xsd:string;
    ];
```

```
    sh:property [
        sh:path schema:email;
        sh:maxCount 1;
        sh:minCount 1;
        sh:nodeKind sh:IRI;
    ].
```

SHACL specify targets using nodes, that must be validated against the shape. There are several types of targets in SHACL:

targetNode directly point to a node. In 2.3, targetNodes are :rifat, :erion and :diago.

targetClass consider all nodes that have a give type. Such as in Shape graph of person in 2.3 targetClass is presented as :Person.

targetProperty defines all nodes that have a given property. It is based on sh:property which is associated with shape property constraint and sh:path identifies the path. An example of property shape present in 2.4.

Listing 2.4 An example of Property Shape

```
@prefix: <http://example.org/>
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@schema schema: <http://schema.org/> .

:PersonShape a sh:NodeShape;
    sh:property [
        sh:path schema:email;
        sh:maxCount 1;
        sh:minCount 1;
        sh:nodeKind sh:IRI;
    ].
```

target node used for SPARQL based general mechanisms.

SHACL shapes uses collection of constraints that nodes must satisfy. Table 2.2 present a list of integrity constraints from the SHACL presentation. In the validation task, a SHACL based approach checks each constraint and reports error for each

unsatisfied constraints. From the SHACL report, if no error found, then we can assume that the RDF graph has been validated. For example, following RDF data graph conforms to the shape graph presented in 2.3. Here target property :rifat passed as it meets the constraints of :name and :email. On the other hand, target property :erion fails as the sh:nodeKind is IRI but it is defined as string in the data graph.

```
# Passed as  :PersonShape
:rifat   schema:name "Mohammad Rashid";
         schema:email <mailto:rifat@gmail.com>

# Fails as :PersonShape
:erion   schema:name "erion cano";
         schema:email "erion@gmail.org"
```

Table 2.2 Integrity constraints components from the SHACL presentation.

Constraints Type	**Example**
Cardinality	minCount, maxCount
Types of values	class, datatype, nodeKind
Values	node, in, hasValue
Range of values	minInclusive, maxInclusive, minExclusive, maxExclusive
String based	minLength, maxLength, pattern, languagesIn, uniqueLanguage
Logical constraints	not, and, or, xone
Closed shapes	closed, ignoredProperties
Property pair constraints	equals, disjoint, lessThan, lessThanOrEquals
Other constraints	name, value, defaultValue

2.4 Knowledge Bases and Their Evolution

A Knowledge Base (KB) is a technology used to store both complex structured and unstructured information's representing domain knowledge[27]. An RDF KB is a well-defined RDF dataset that consists of RDF statements (triples) of the form (*subject*, *predicate*, *object*). Considering description logic (DL), a knowledge base is the equivalent of a theory in first-order logic or an ontology in OWL. A DL

knowledge base is based on a set of terminological axioms (TBox) and assertions axioms (ABox) [28]. Axioms are statements that are asserted to be true in the domain being described[14]. In the LOD cloud, KBs are created in various ways and depend on specific domain [29]. For example, proprietary and curated KB like 3cixty [30] or the crowd-based KB like Freebase [31] and Wikidata [27]. Furthermore, KB can be based on data extracted from large-scale, semi-structured sources such as DBpedia [32] using Wikipedia.

2.4.1 Use Cases: 3cixty and DBpedia

In our approach we explored two KBs as use cases namely, 3cixty Nice KB and DBpedia KB. We selected 3cixty Nice KB and DBpedia KB according to: *(i)* popularity and representativeness in their domain: DBpedia for the encyclopedic domain, 3cixty Nice for the tourist and cultural domain; *(ii)* heterogeneity in terms of content being hosted such as periodic extraction of various event information extracted by 3cixty Nice KB, *(iii)* diversity in the update strategy: incremental and usually as batch for DBpedia, continuous update for 3cixty. More in detail:

- *DBpedia*[15] is among the most popular knowledge bases in the LOD cloud. This knowledge base is the output of the DBpedia project that was initiated by researchers from the Free University of Berlin and the University of Leipzig, in collaboration with OpenLink Software. DBpedia is roughly updated every year since the first public release in 2007. DBpedia is created from automatically extracted structured information contained in Wikipedia, such as infobox tables categorization information, geo-coordinates, and external links.

- *3cixty Nice* is a knowledge base that describes cultural and tourist information. This knowledge base was initially developed within the 3cixty project[16], which aimed to develop a semantic web platform to build real-world and comprehensive knowledge bases in the domain of culture and tourism for a few cities. The entire approach has been tested first in the occasion of the Expo Milano 2015 [25], where a specific knowledge base for the city of Milan was developed, and has now been refined with the development of knowledge bases

[14] https://www.w3.org/TR/2002/WD-owl-semantics-20021108/syntax.html
[15] http://wiki.dbpedia.org
[16] https://www.3cixty.com

for the cities of Nice, London, Singapore, and Madeira island. They contain descriptions of events, places (sights and businesses), transportation facilities and social activities, collected from numerous static, near- and real-time local and global data providers, including Expo Milano 2015 official services in the case of Milan, and numerous social media platforms. The generation of each city-driven 3cixty KB follows a strict data integration pipeline, that ranges from the definition of the data model, the selection of the primary sources used to populate the knowledge base, till the data reconciliation used for generating the final stream of cleaned data that is then presented to the users via multi-platform user interfaces. The quality of the data is today enforced through a continuous integration system that only verifies the integrity of the data semantics [30].

2.4.2 Knowledge Base Evolution

Knowledge bases are nowadays essential components for any task that requires automation with some degrees of intelligence. Furthermore, KBs are performing an important role in the organizations data management and in supporting data integration. Stakeholders – curator, consumer, etc. – in various domains routinely need to combine and compare statistical indicators for various applications. The majority of the knowledge bases are domain specific and maintained by small groups of knowledge engineers. Moreover, updating the knowledge bases become very cost intensive as data sources evolves and need to consolidate those changes with each release. Within the context of KB evolution, aspects of the KB evolution managements are becoming increasingly important to produce the data consistent and usable.

For example, Wikipedia has grown into one of the central hubs of knowledge sources, and it is maintained by thousands of contributors. DBpedia is a crowd-sourced knowledge base and extracts structured information from various Wikimedia projects. Table 2.3 illustrates the overall statistics of English DBpedia[17] on three different releases. With each DBpedia release, the raw infobox statements grow about 4.2%. Furthermore, millions of triples have been included with each version

[17] http://wiki.dbpedia.org/

Table 2.3 Statistics of the English DBpedia KB updates on the three releases of 201504, 201510 and 201604.

Features	**201504**	**201510**	**201604**
Localiced Instances	4,305,028	4,641,890	4,678,230
Canonicalized Instances	4,305,028	4,641,890	4,678,230
Mapping based Properties	1,353	1,369	1,379
Mapping based Statements	32,143,157	37,852,643	37,549,405
Raw Infobox Properties	58,780	60,461	2,062
Raw Infobox Statements	73,686,499	78,498,880	30,024,092
Type Statements	34,911,097	36,583,668	36,704,825

of DBpedia released. For any data stakeholder managing, this exponential growth of information is a challenging task.

Figure 2.1 illustrates common use cases [10] of knowledge base evolution. These use cases are explained in detail below.

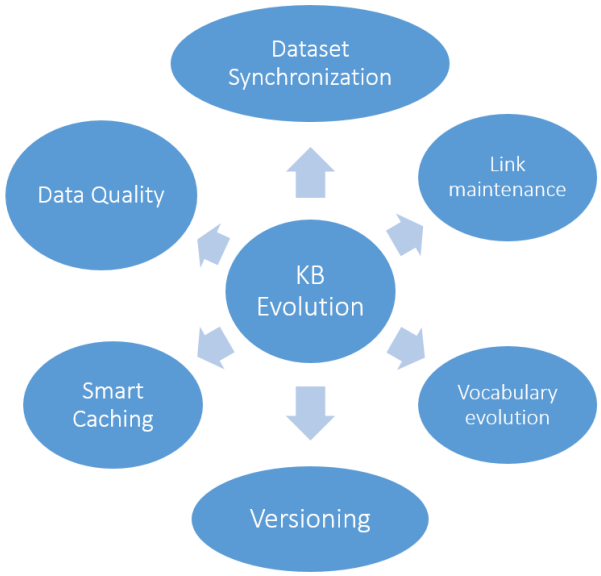


Fig. 2.1 Use cases of Knowledge Base evolution.

- *Dataset Synchronization:* In any KB, large quantity of data needs to be replicated and maintained at external sources. Furthermore, these data sources need to be in periodic synchronization with the original data sources [10].

- *Link maintenance:* In a KB, data is made of statements that link between resources. Due to KB updates, often resources are erroneously removed or change semantics, without taking the necessary steps to update their dependent resources. Thus, it creates the need to take appropriate action for link maintenance.

- *Vocabulary evolution:* Ontologies, vocabularies, and data schemata in a KB are often inconsistent and lack metadata information. Due to ontology evolution, it is difficult to find practical guidelines and best practices [20].

- *Versioning:* It is relevant for ontologies, vocabularies, and data schemata in a KB, whose semantics may change over time to reflect usage [10]. Within this context, KB evolution analysis can show how changes propagate and help to design a stable versioning methodology.

- *Smart Cashing:* Query optimization and live querying approaches need a smart cashing approach for dereferencing and sources discovery. KB evolution analysis can help to identify which sources can be cached to save time and resources, how long cached data can be expected to remain valid, and whether there are dependencies in the cache [10].

- *Data Quality:* One of the key use case is to ensure a good quality of data in a KB. Since data instances are often derived from autonomous, evolving, and increasingly large data providers, it is impractical to do manual data curation, and at the same time, it is very challenging to do the continuous automatic assessment of data quality. In this context, using the KB evolution analysis, we can explore the data quality issues.

2.5 Data Quality Standards

Data quality can be defined as the degree to which a set of characteristics of data fulfills requirements [1]. Based on user requirements, poor data quality can be defined as the degree to which a set of characteristics of data does not fulfill the requirements [33]. The definition of our proposed quality characteristics started with the exploration of two data quality standard reference frameworks: ISO/IEC 25012 [1] and W3C DQV [21].

Table 2.4 Measurement terminology

Definition	ISO 25012	W3C DQV
Category of quality attributes	Characteristic	Dimension
Variable to which a value is assigned as the result of a measurement function applied to two or more measure elements	Measure	Metric
Variable defined in terms of an attribute and the elements for quantify the measurement method	Measure Element	-
Quality measurement results that that characterize a quality feature	Numerical Value	Observation
Set of operations having the object of determining a value of a measure	Measurement	Measurement

ISO/IEC 25012 [1] defines a general data quality model for data retained in structured format within a computer system. This model defines the quality of a data product as the degree to which data satisfies the requirements set by the product owner organization. The W3C Data on the Web Best Practices Working Group has been chartered to create a vocabulary for expressing data quality[1]. The Data Quality Vocabulary (DQV) is an extension of the DCAT vocabulary[18]. It covers the quality of the data, how frequently it is updated, whether it accepts user corrections, and persistence commitments.

Besides, to further compare our proposed quality characteristics[19] we explored the foundational work on the linked data quality by Zaveri *et al.* [22]. They surveyed existing literature and identified a total of 26 different data quality dimensions (criteria) applicable to linked data quality assessment.

Since the measurement terminology suggested in these two standards differs, we briefly summarize the one adopted in this paper and the relative mappings in Table 2.4.

[18] https://www.w3.org/TR/vocab-dcat/

[19] In our work we will identify the quality aspects using the term quality characteristics from ISO-25012 [1] that corresponds to the term quality dimension from DQV [21].

2.6 Data Quality Issues in Evolving KB

A data quality issue is a set of anomalies that can affect the knowledge base exploitation and any application usage [34]. We can identify quality issues through data quality measurement. A data quality characteristic analyzes quality issues by using a set of measurement functions that help to assess the issues present in the data [35]. In this thesis, we focused on three main quality issues of a knowledge base: *(i)* Lack of consistency, *(ii)* Lack of completeness, and *(iii)* Lack of persistency.

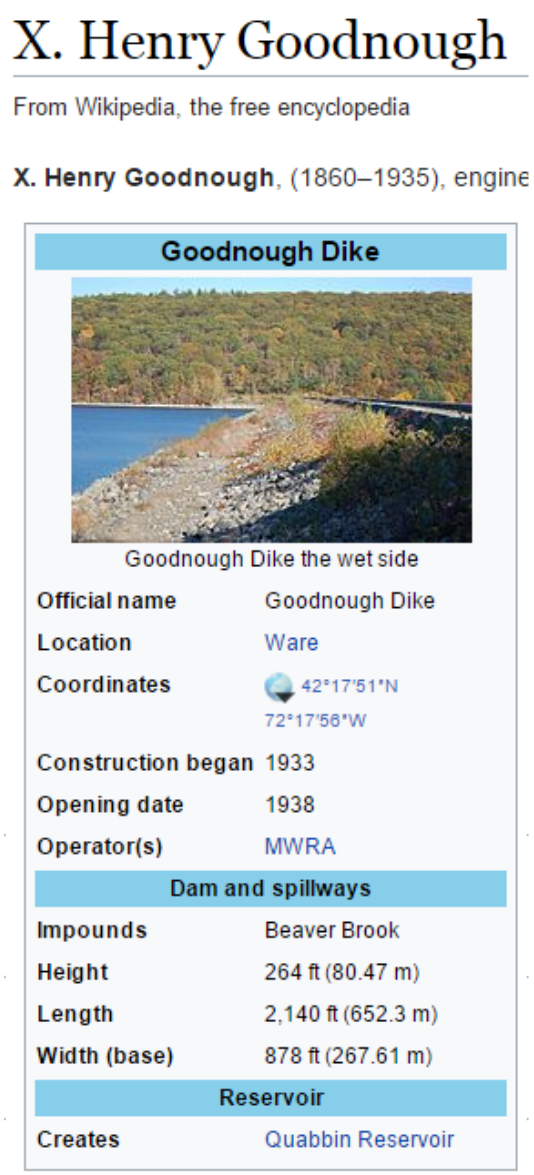
Lack of consistency when a KB is inconsistent with the reality it represents. In particular, inconsistency relates to the presence of unexpected properties.

As an example, let us consider DBpedia version 201510 where we can find the resource of type *foaf:Person dbpedia:X. Henry Goodnough* that represent an entity. While we find (as expected) a *dbo:birthDate* property for the entity, we unexpectedly find the property *dbo:Infrastructure/length*. This is a clear inconsistency: in fact, if we look at the ontology, we can check that the latter property can be used for a resource of type *dbo:Infrastructure*, not for a person.

To better understand where the problem lies, we need to look at the corresponding Wikipedia page wikipedia-en:X._Henry_Goodnough[20]. Even though the page reports the information about an engineer who graduated from Harvard, it contains an info-box, shown in Figure 2.2, that refers to a dam, the Goodnough Dike. The inconsistency issue derives from the data present in the source page that resulted into the resource being typed both as a person and as a piece of infrastructure. We can expect such kind of structure to be fairly rare – in fact the case we described is the only case of a person with a *dbo:Infrastructure/length* property – and can be potentially detected by looking at the frequency of the predicates within a type of resource. For instance, considering DBpedia version 2016-04, for the resources of type *foaf:Person* there are 1035 distinct predicates, among which 142 occur only once. Such anomalous predicates suggests the presence of consistency issues that can be located either in the original data source or – i.e. Wikipedia for this case – or in the lack of filtering in the data extraction procedure.

Lack of completeness relates to the resources or properties missing from a knowledge base. This happens when information is missing from one version of the

[20] https://en.wikipedia.org/wiki/X._Henry_Goodnough



X. Henry Goodnough

From Wikipedia, the free encyclopedia

X. Henry Goodnough, (1860–1935), engine

Goodnough Dike	
Goodnough Dike the wet side	
Official name	Goodnough Dike
Location	Ware
Coordinates	42°17'51"N 72°17'56"W
Construction began	1933
Opening date	1938
Operator(s)	MWRA
Dam and spillways	
Impounds	Beaver Brook
Height	264 ft (80.47 m)
Length	2,140 ft (652.3 m)
Width (base)	878 ft (267.61 m)
Reservoir	
Creates	Quabbin Reservoir

Fig. 2.2 Example of inconsistent Wikipedia data (December 2016).

KB because it has been removed at given point during KB's evolution[21]. In general, causes of completeness issues are linked to errors in the data extraction pipeline. Such as missing instances in a KB that are auto generated from data sources. As an example, let us consider a DBpedia resource *dbpedia:Abdul_Ahad_Mohma* of type *dbo:Person/Astronauts*[22]. When looking at the source Wikipedia page *wikipedia-*

[21] Of course it is possible the item was never present in the KB at any time during its evolution, though this kind of mistake is not detectable just by looking at the evolution of the KB.

[22] http://dbpedia.org/resource/Abdul_Ahad_Mohmand

en:Abdul_Ahad_Mohman[23], we observe that the infobox shown in Figure 2.3 reports a "Time in space" datum. The DBpedia ontology includes a *dbo:Astronaut/TimeInSpace* and several other astronauts have that property, but the resource we consider is missing it.

Fig. 2.3 Example of incomplete Wikipedia data (December 2016).

While it is generally difficult to spot that kind of incompleteness, for the case under consideration it is easier because that property was present for the resource present in the previous version of DBpedia, i.e. the 2015-10 release. That is an incompleteness introduced by the evolution of the knowledge base. It can be spotted by looking at the frequency of predicates inside a resource type. In particular, in the release of 2016-04 there are 419 occurrences of the *dbo:Astronaut/TimeInSpace* predicate over 634 astronaut resources (66%), while in the previous version there were 465 out of 650 astronauts (72%). Such a significant variation suggests the presence of a major problem in the data extraction procedure applied to the original source, i.e. Wikipedia.

Lack of persistency relates to unwanted removal of persistent resources that were present in a previous KB release but they disappeared. This happens when information has been removed. As an example let us consider a 3cixty Nice re-

[23] https://en.wikipedia.org/wiki/Abdul_Ahad_Mohmand

source of type *lode:Event* that has as the label "Modéliser, piloter et valoriser les actifs des collectivités et d'un terrritoire grâce aux maquettes numériques: retours d'expériences et bonnes pratiques"[24]. This resource happened to be part of the 3cixty Nice KB since it has been created the first time, but in a release it got removed even though, according to the experts curating the KB, it should not have been removed.

Subject Item
n2:006dc982-15ed-47c3-bf6a-a141095a5850
rdf:type
lode:Event
rdfs:label
Modéliser, piloter et valoriser les actifs des collectivités et d'un terrritoire grâce aux maquettes numériques : retours d'expériences et bonnes pratiques
rdfs:seeAlso
n13:en
cixty:descriptionScore
0.0
cixty:posterScore
1.0
lode:poster
n4:006dc982-15ed-47c3-bf6a-a141095a5850
dc:identifier
MN13
dc:publisher
n14:com
locationOnt:businessType
n15:event
lode:atPlace
n12:be7fac75-bb59-41fd-a626-4bd7e77f0a7f
lode:atTime
n6:interval
lode:hasCategory
Conferences Maquette Numérique
lode:inSpace
n6:geometry
lode:involvedAgent
n11:a40c9900f85a517cef40ef8f1e4289b9 n11:7f1a9cc96861920e147505e23ea4f913
n11:dce31cbcfdad5c0a180fb4d0efd0c511
locationOnt:cell
n9:1301

Fig. 2.4 Example of a 3cixty Nice KB resource that unexpectedly disappeared from the release of 2016-06-15 to the other 2016-09-09.

This issue can be spotted by looking at the total frequency of entities of a given resource type. For example, *lode:Event* type two releases – 2016-06-15 and 2016-09-09 – total entity count 2,182 and 689. In particular in the investigated example we have observed an (unexpected) drop of resources of the type *event* between the previous release dated as 2016-06-15 and the considered released from 2016-09-09. Such count drop actually indicates a problem in the processing and integration of the primary sources that feed the KB.

[24] http://data.linkedevents.org/event/006dc982-15ed-47c3-bf6a-a141095a5850

Such problems are generally complex to be traced manually because they require a per-resource check over different releases. When possible, a detailed, low-level and automated analysis is computationally expensive and might result into a huge number of fine-grained issue notifications. Such amount of information might cause an information overload for the user of the notifications. However, provided they are filtered, such low-level notifications can be useful to KB end-users to assess the suitability for their purposes.

Such a problem is generally complex to be traced manually because it requires a per-resource check over the different releases. It can, instead, be spotted by looking at the total frequency of entities of a given resource type.

2.7 Gold Standard Creation

KBs may contain errors, thus the profiling results cannot be considered as a gold standard. To instruct and verify the performance of the modeling process, the feature datasets are further validated by a human annotator. More specifically, to reduce the wrong annotations a manual validation is performed by a human annotator. There are different strategies to evaluate a dataset. In the following, we present three common strategies when dealing with a knowledge base [29]:

(i) Silver Standard: this strategy is based on the assumption that the given KB is already of reasonable quality. The silver standard method is usually applied to measure the performance of knowledge graph by analyzing how well relations in a knowledge graph can be replicated. Although this strategy is suitable for large-scale data, it can produce less reliable results [36, 37].

(ii) Gold standard: this strategy is based on turning the observations in a set of gold data points by human annotators. In this context, gold standard is indeed suitable for our approach since we can obtain gold insights of the completeness measurement results, however very expensive if the annotation load is large.

(iii) Partial gold standard: in this strategy, a small subset of external graphs, entities or relations are selected as validation criteria and they, then, are manually labeled [36]. This helps reducing the number of candidates that an annotator will process.

2.8 Learning Models

One of the goals of this thesis is to validate the RDF KBs using integrity constraints in predictive settings. In this context, we present our approach as a classification problem. Typically a classification learning model maps observations (samples) to a set of possible categories (classes) [38]. For example, the minimum cardinality value of an entity type is an observation for its relevant attributes (features). For selecting a suitable learning model for our problem, we investigated the following research question: *"Which learning model is the most adequate for consistency analysis using data profiling information as predictive features?"*. In order to answer this question, we evaluate the performance of predictive features using five classical learning models. These learning models are chosen considering five popular categories of machine learning algorithms [38]: *(i)* Neural Networks, *(ii)* Bayesian, *(iii)* Instance Based, *(iv)* Support Vector Machine, and *(v)* Ensemble. Following we present details of five well-known learning models.

***Multilayer Perceptron* [39]:** a feed forward Neural Network consisting of at least three layers of neurons with a non-linear activation function: one for inputs, one for outputs and one or more hidden layers. Training is carried out through back propagation.

Naive Bayes [40]: is a simple probabilistic classifier. The core concept is based on the Bayes theorem [40]. Generally, naive bayes classifiers are based on the assumption that features are independent with each other.

k-Nearest Neighbors (k-NN) [41]: is an instance-based learning algorithm. It locates the k-nearest instances to the input instance and determines its class by identifying the single most frequent class label. It is generally considered not tolerant to noise and missing values. Nevertheless, it is highly accurate, insensitive to outliers and works well with both nominal and numerical features.

Support Vector Machines (SVM) [42]: it conceptually implements the following idea: input vectors are non-linearly mapped to a very high dimensional feature space. In this feature space a linear decision surface is constructed. Special properties of the decision surface ensures high generalization ability of the classifier.

Random Forest [43]: it creates many classification trees. To classify a new object from an input vector, it maps the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

In our modeling phase we applied *k-fold cross validation* [38] to reduce the variance of a performance score. In the *k-fold cross validation*, k is the number of splits to make in the dataset. In this approach, we choose value of k=10. This will result in splitting the dataset into 10 portions (10 folds) and run the learning model 10 times. For each algorithm training will be run on 90% of the data and testing on the 10%. The model will exchange which 10% of the data is tested with each run. In particular with k value of 10 will use each data instance as a training instance exactly 9 time and test instance 1 time.

We also adopted general classification performance evaluation measures such as precision, recall, and F1 score [44]. Evaluation of the classification performance is based on considering one of the output classes as the positive class and defining: *(i)* true positives (TP): the number of samples correctly labeled as in the positive class; *(ii)* false positives (FP): the number of samples incorrectly labeled as in the positive class; *(iii)* true negatives (TN): the number of samples correctly labeled as not in the positive class; *(iv)* false negatives (FN): the number of samples incorrectly labeled as not in the positive class.

We present the formulas of the aforementioned metrics:

Precision (P): It is based on positive predictive value, defined as $P = \frac{TP}{TP+FP}$;

Recall (R): its related to true positive rate also know as sensitivity, defined as $R = \frac{TP}{TP+FN}$;

F1 Score ($F1$): its a measure of test accuracy and defined as the harmonic mean of precision and recall: $F1 = \frac{2*P*R}{P+R}$.

Chapter 3

State of the Art

This chapter provides an overview over the state-of-the art in the context of Knowledge Base (KB) quality assessment and validation approaches. The research activities related to this thesis fall into three main areas: (i) Linked Data Dynamics, (ii) Knowledge Base Quality Assessment, and (iii) Knowledge Base Validation. In Section 3.1, we illustrate the related works regarding linked data dynamics as well as the main challenges present in this research area. Next, in Section 3.2 we present a comparative survey of quality assessment for evolving KBs. Finally, Section 3.3 reports a detailed overview of the existing approaches for KB validation.

3.1 Linked Data Dynamics

Taking into account changes over time, every dataset can be dynamic. In this context, a comparative analysis is present by Umbrich *et al.* [45]. More specifically, they analyzed entity dynamics using a labeled directed graph based on LOD, where a node is an entity that is represented by a subject. In addition, Umbrich *et al.* [46] present a comprehensive survey based on technical solutions for dealing with changes in datasets of the Web of Data.

Considering dataset dynamicity, Käfer *et al.* [10] design a Linked Data Observatory to monitor linked data dynamics. They setup a long-term experiment to monitor the two-hop neighbourhood of a core set of eighty thousand diverse Linked Data documents on a weekly basis. They explore the dataset dynamics taking into consid-

eration of five use cases such as synchronization, smart caching, hybrid architectures, external-link maintenance, and vocabulary evolution and versioning.

Papavasileiou *et al.* [8] explored high-level change detection in RDF(S) KBs by addressing change management for RDF(S). They target the data management issues in KBs where data is maintained by large communities, such as scientists or librarians, who act as curators to ensure high quality of data. Such curated KBs are constantly evolving for various reasons, such as the inclusion of new experimental evidence or observations, or the correction of erroneous conceptualizations. Managing such changes poses several research problems, including the problem of detecting the changes (delta) among versions of the same KB developed and maintained by different groups of curators, a crucial task for assisting them in understanding the involved changes. They addressed this problem by proposing a language change that allows the formulation of concise and intuitive deltas. Similarly, in our work, we explore the deltas present in consecutive KB releases using data profiling.

Pernelle *et al.* [15] present an approach that detects and semantically represents data changes in RDF datasets. Klein *et al.* [47] analyze ontology versioning in the context of the Web. They look at the characteristics of the release relation between ontologies and at the identification of online ontologies. Then they describe a web-based system to help users to manage changes in ontologies.

In [20], Mihindukulasooriya *et al.* present an empirical analysis of the ontologies that were developed in a collaborative manner to understand community-driven ontology evolution in practice. They have analyzed, how four well-known ontologies (DBpedia, Schema.org, PROV-O, and FOAF) have evolved through their lifetime and they observed that quality issues were due to the ontology evolution. They pointed out the need of having multiple methodologies for managing changes. They also summarize that the selected ontologies do not follow the theoretical frameworks found in literature. They present that the most common quality problems caused by ontology changes include the use of abandoned classes and properties in data and the presence of duplicate classes and properties. Nevertheless, their work is not focused on dataset changes but rather how changes in the ontology affect the data described using those ontologies.

In [7], Nishioka *et al.* present a clustering technique over the dynamics of entities to determine common temporal patterns. The quality of the clustering is evaluated using entity features such as the entities' properties, RDF types, and

pay-level domain. In addition, they investigated to what extent entities that share a feature value change over time. In this work, we explore dynamic features for detecting completeness issues. Instead of using a clustering technique [7] based on the temporal pattern of entities, we focus on presenting the evolution analysis as a classification problem to detect quality issues.

3.2 Knowledge Base Quality Assessment

In the field of Linked Data, quality is a largely investigated research field, and many approaches to data quality management have been proposed. There exists a large number of data quality frameworks and tools based on manual, crowd-sourced, and automatic approaches. The main contribution of this thesis is a quality assessment approach for evolving KB using data profiling. We consider data profiling as a key measurement component for our approach. In this section, we review literature that analyze the quality of various aspects of KBs.

Comprehensive Studies. A comprehensive overview of the RDF dataset profiling presented by Ellefi *et al.* [9]. Authors explored the RDF dataset profiling feature, methods, tools, and vocabularies. They present dataset profiling in a taxonomy and illustrate the links between the dataset profiling and feature extraction approaches. They organized dataset profiling features into seven top-level categories: 1. General; 2. Qualitative; 3. Provenance; 4. Links; 5. Licensing; 6. Statistical; 7. Dynamics. They considered dataset dynamics as profiling features using the study presented by Käfer *et al.* [10]. Similarly, in this work, we explored the concepts regarding qualitative, statistical, and dynamic features. Furthermore, we also explored the studies regarding RDF data validation languages. In this work, we explored the dataset dynamics features using periodic data profiling. More specifically, we explored the aspects of dataset dynamics introduced by Ellefi *et al.* [9] to analyse various quality characteristics.

Considering the data quality methodologies applied to linked open data (LOD), a comprehensive systematic literature review presented by Zaveri *et al.* [22]. They have extracted 26 quality dimensions and a total of 110 objective and subjective quality indicators. They organized linked data quality dimensions into following categories, 1. Contextual dimensions; 2. Trust dimensions; 3. Intrinsic dimensions; 4. Accessibility dimensions; 5. Representational dimensions; 6. Dataset dynamicity

dimensions. They explored dataset dynamics features based on three dimensions: 1. Currency: speed of information update regarding information changes; 2. Volatility: length of time which the data remains valid; 3. Timeliness: information is available in time to be useful. The work presented in this thesis is related to intrinsic, contextual and dataset dynamicity dimensions.

Quality Assessment Frameworks. Taking into account data quality analysis using manual approaches, Bizer *et al.* [48] present WIQA, a quality assessment framework. This framework enables information consumers to apply a wide range of policies to filter information. It employs the Named Graphs data model for the representation of information together with quality-related meta-information and uses the WIQA-PL[1] policy language for expressing information filtering policies. WIQA-PL policies are expressed in the form of graph patterns and filter conditions. WIQA can be used to understand the intended changes present in a KB by applying graph patterns and filtering conditions. Based on this study, instead of a static version of a KB, we can explore multiple versions of KB using WIQA policy.

Using provenance metadata information, Mendes *et al.* [49] present Sieve framework that uses user configurable quality specification for quality assessment and fusion method. Sieve is integrated as a component of the Linked Data Integration Framework (LDIF).[2] In particular, Sieve uses the LDIF provenance metadata and the user configured quality metrics to generate quality assessment scores. They propose a set of Linked Data quality assessment measures such as: 1. Intensional completeness; 2. Extensional completeness; 3. Recency and reputation; 4. Time since data modification; 5. Property completeness; 6. Property conciseness; 7. Property consistency. Instead of LDIF provenance metadata and user configuration, we focus on the dataset dynamics features from data profiling.

In [50], Kontokostas *et al.* propose a methodology for test-driven quality assessment of Linked Data. They formalize quality issues and employ SPARQL query templates, which are instantiated into quality test queries. Using test driven quality assessment approach, they present RDFUnit[3] a tool centered around schema validation. It runs automatically based on a schema and manually generates test cases against an endpoint. RDFUnit has a component that turns RDFS axioms and simple OWL axioms into SPARQL queries that check for data that does not match

[1] http://wifo5-03.informatik.uni-mannheim.de/bizer/wiqa/
[2] http://ldif.wbsg.de/
[3] https://github.com/AKSW/RDFUnit

the axiom. In contrast, in this study, we aim to learn the constraints (which might not be explicitly stated as RDFS or OWL axioms) as RDF Shapes. Although the overall objectives are somewhat similar to this work, for quality analysis we mainly explore aspects of KB dynamicity. Furthermore, in our approach, we mainly use data profiling information as the input for the process. Results from the consistency analysis can be extended by using RDFUnit for further validation.

In [35], Debattista *et al.* describes a conceptual methodology for assessing Linked Datasets, proposing Luzzu, a framework for Linked Data Quality Assessment. Luzzu is based on four major components: 1. An extensible interface for defining new quality metrics; 2. An interoperable, ontology-driven back-end for representing quality metadata and quality problems that can be re-used within different semantic frameworks; 3. Scalable dataset processors for data dumps, SPARQL endpoints, and big data infrastructures; 4. A customisable ranking algorithm taking into account user-defined weights. Luzzu is a stream-oriented quality assessment framework that focuses on data instance-centric measurement of a user-defined collection of quality metrics. The metrics validation requires users to write Jave code for implementing some checks. Resulting to be not feasible for many users such as knowledge engineers since they need to write an OWL reasoner to detect the logical errors in the unified dependency tree of a linked data schema. Furthermore, various research works explored the importance of quality metrics in a probabilistic and deterministic settings. Debattista *et al.* [51], explored probabilistic techniques such as Reservoir Sampling, Bloom Filters and Clustering Coefficient estimation for implementing a broad set of data quality metrics in an approximate but sufficiently accurate way. On the other hand, various research works put emphasis on the problem of error detection in a KB. For example, distance-based outlier detection by Debattista *et al.* [52] and error detection in relation assertions by Melo *et al.* [53] gave more focus towards error detection in schemes. Although the core concepts are somewhat similar to this work, the focus of this study is on dataset dynamics and detection of quality issues using multiple versions of the same dataset. We explore the persistency, completeness and consistency issues by using data profiling for evolving KBs.

Crowdsourcing. To understand the intended changes by stakeholders due to KB updates, a crowd-sourcing quality assessment approach can be used. A crowd-sourcing quality assessment approach has been introduced by Acosta *et al.* [54] for quality issues that are difficult to uncover automatically. They explore most common quality issues in DBpedia datasets, such as incorrect object values, incorrect datatype

or language tag and incorrect link. The authors introduce a methodology to adjust crowdsourcing input from two types of audience: *(i)* Linked Data experts through a contest to detect and classify erroneous RDF triples and *(ii)* Crowdsourcing through the Amazon Mechanical Turk. In detail, they adapt the Find-Fix-Verify crowdsourcing pattern to exploit the strengths of experts and paid workers. Furthermore, they use TripleCheckMate [55] a crowdsourcing tool for the evaluation of a large number of individual resources, according to a defined quality problem taxonomy. To understand the quality of data sources, Flemming's [56] present an assessment tool that calculates data quality scores based on manual user input for data sources. More specifically, a user needs to answer a series of questions regarding the dataset and assigns weights to the predefined quality metrics. However, it lacks several quality dimensions such as completeness or inconsistency.

Metadata. In [57], Assaf *et al.* introduce a framework that handles issues related to incomplete and inconsistent metadata quality. They propose a scalable automatic approach for extracting, validating, correcting and generating descriptive linked dataset profiles. This approach applies several techniques in order to check the validity of the metadata provided and to generate descriptive and statistical information for a particular dataset or for an entire data portal. In particular, they extensively used dataset metadata against an aggregated standard set of information. This leads to dependency towards availability of metadata information. Instead, in our approach, we only focus on summary statistics from collected dataset, and it is independent from external information since the quality profiling can be done only using summary statistics.

Temporal Analysis. In [58], Rula *et al.* start from the premise of dynamicity of Linked Data and focus on the assessment of timeliness in order to reduce errors related to outdated data. To measure timeliness, they define a currency metric that is calculated in terms of differences between when the observation is done (current time) and the time when the data was modified for the last time. Furthermore they also take into account the difference between the time of data observation and the time of data creation. Similary, in our work, we explore KB dynamicity using data profiling information. Rather than using timeless measures, we explore the change behaviour present in the dataset using dynamic profiling features introduced by Ellefi *et al.* [9].

In [59], Furber and Hepp focus on the assessment of accuracy, which includes both syntactic and semantic accuracy, timeliness, completeness, and uniqueness. One measure of accuracy consists of determining inaccurate values using functional dependence rules, while timeliness is measured with time validity intervals of instances and their expiry dates. Completeness deals with the assessment of the completeness of schema (representation of ontology elements), completeness of properties (represented by mandatory property and literal value rules), and completeness of population (representation of real world entities). Uniqueness refers to the assessment of redundancy, i.e., of duplicated instances. In this work, we explored the changes present in the KB to identify quality issues.

Considering the version management and linked data lifecycle, Knuth *et al.* [60] identify the key challenges for Linked Data quality. As one of the key factors for Linked Data quality they outline validation that, in their opinion, has to be an integral part of Linked Data lifecycle. Additional factor for Linked Data quality is version management, which can create problems in provenance and tracking. Finally, as another important factor they outline the usage of popular vocabularies or manual creating of new correct vocabularies. Futhermore, Emburi *et al.* [61] developed a framework for automatic crawling the Linked Data datasets and improving dataset quality. In their work, the quality is focused on errors in data and purpose of the developed framework is to automatically correct errors.

Statistical analysis. Paulheim *et al.* [62] present two approaches SDType and SDValidate for quality assessment. SDType predicts classes of RDF resources thus completing missing values of rdf:type properties. SDValidate detects incorrect links between resources within a dataset. These methods can effectively detect errors on DBpedia, however they require the existence of informative type assertions. Furthermore, more complex errors containing wrong entities with correct types cannot be detected. Taking into account probabilistic approach for linked data quality assessment, Li *et al.* [63] presented a probabilistic framework using the relations (equal, greater than, less than) among multiple RDF predicates to detect inconsistencies in numerical and date values based on the statistical distribution of predicates and objects in RDF datasets. However, they mainly focused on detecting errors in the numerical data. In [64], Ruckhaus *et al.* present LiQuate, a tool based on probabilistic models to analyze the quality of data and links. They use Bayesian Networks and rule-based system for quality assessment. The probabilistic rules are represented by data experts to identify redundant, incomplete and inconsistent links

in a set of resources. In our approach, we mainly focus on the statistical profiling at instance level. This reduces the dependency towards expert intervention.

In the current state of the art, less focus has been given toward understanding knowledge base resource changes over time to detect anomalies and completeness issues due to the KB evolution. Furthermore, for an evolving KB, we investigated two perspectives: *(i)* Static: data quality analysis with respect to a specific tasks without considering dataset dynamics; *(ii)* Dynamic: process of accessing data and temporal analysis such as timeliness measure. In Table 3.1, we summarize the reported linked data quality assessment approaches.

3.3 Knowledge Base Validation

The problem of RDF data validation has been researched using *Description Logics* considering both Open World and Closed World Assumption. The Web Ontology Language (OWL) [65] is an expressive ontology language based on Description Logics (DL). The semantics of OWL addresses distributed knowledge representation scenarios where complete knowledge about the domain cannot be assumed. Motik *et al.* [66] proposed an extension of OWL that attempts to mimic the intuition behind integrity constraints in relational databases. They divided axioms into regular axioms and constraints. To address the above mismatch some approaches use OWL expressions with Closed World Assumption and a weak Unique Name Assumption so that OWL expressions can be used for validation purposes such as Stardog ICV[4] and Tao *et al.* [67].

Generally, Description Logics (DLs), in turn, bear a first-order predicate logic semantics. DLs are monotonic and adhere to the Open World Assumption (OWA). This means that negative or positive conclusions drawn from a knowledge base must be based on information explicitly present in a knowledge base. Therefore, negative conclusion may lead to possible logical issues. Under the Closed World Assumption (CWA) all non-provable expressions are assumed to be false [68]. In [68], Patel-Schneider explored Description Logics as a mean to provide the necessary framework for both checking constraints and providing CWA facilities. They utilized inference as a mean for constraint checking, which is the core service provided by Description

[4] https://www.stardog.com/docs/

Table 3.1 Summary of Linked Data Quality Assessment Approaches.

Paper	Degree of Automation	Goal	Dataset Feature
Bizer *et al.* [48]	Manual	WIQA quality assessment framework enables information consumers to apply a wide range of policies to filter information.	Static
Acosta *et al.* [54]	Manual	A crowd-sourcing quality assessment approach for quality issues that are difficult to uncover automatically.	Static
Ruckhaus *et al.* [64]	Semi-Automatic	LiQuate, a tool based on probabilistic models to analyze the quality of data and links.	Static
Paulheim *et al.* [62]	Semi-Automatic	SDType approach using statistical analysis to predicts classes of RDF resources thus completing missing values of rdf:type properties.	Static
Furber and Hepp [59]	Semi-Automatic	Focus on the assessment of accuracy, which includes both syntactic and semantic accuracy, timeliness, completeness, and uniqueness.	Dynamics (Timeliness analysis)
Flemming [56]	Semi-Automatic	Focuses on a number of measures for assessing the quality of Linked Data covering wide-range of different dimensions such as availability, accessibility, scalability, licensing, vocabulary reuse, and multilingualism.	Static
Mendes *et al.* [49]	Semi-Automatic	Sieve framework that uses user configurable quality specification for quality assessment and fusion method.	Dynamic (Timeliness analysis)
Knuth *et al.* [60]	Semi-Automatic	They outline validation which, in their opinion, has to be an integral part of Linked Data lifecycle.	Static
Rula *et al.* [58]	Automatic	Start from the premise of dynamicity of Linked Data and focus on assessment of timeliness in order to reduce errors related to outdated data.	Dynamic (Timeliness analysis)
Kontokostas *et al.* [50]	Automatic	Propose a methodology for test-driven quality assessment of Linked Data.	Dynamic
Emburi *et al.* [61]	Automatic	They developed a framework for automatic crawling the Linked Data datasets and improving dataset quality.	Dynamic (Temporal Analysis)
Li *et al.* [63]	Automatic	They proposed an automatic method to detect error between multi attributes which can not be detected only considering single attribute.	Dynamic
Assaf *et al.* [57]	Automatic	They propose a framework that handles issues related to incomplete and inconsistent metadata quality.	Static
Debattista *et al.* [35]	Automatic	They propose a conceptual methodology for assessing Linked Datasets, proposing Luzzu, a framework for Linked Data Quality Assessment.	Static

Logics. OWA makes it challenging to perform certain validation tasks. For example, a minimum cardinality constraint cannot be violated under OWA because there is always a possibility that a triple exists somewhere. Furthermore, under certain circumstances reasoners can find some inconsistencies using the axioms present in OWL model. This utility can lead to a confusion to think of ontological languages as validation languages. Nevertheless, the underpinning principles used in OWL such as the use of Open World Assumption (OWA) and Non-Unique Name Assumption, can lead to unexpected and confusing results in a validator [68, 69].

Ontology based learning is commonly defined as a field that comprises techniques for automated acquisition of ontological knowledge from data. Thus, the paradigm has shifted such that many approaches do not aim to generate a full fledged, gold-standard ontology from data anymore, but they rather focus on acquiring axioms of certain shapes such as concept definitions, atomic subsumptions, disjointness axioms. There are several works done on induction of Description Logic axioms using methods, such as:

Association rule mining (ARM). Abedjan *et al.* [70] present rule-based approaches for predicate suggestion, data enrichment, ontology improvement, and query relaxation. They identified inconsistencies in the data through predicate suggestion, enrichment with missing facts, and alignment of the corresponding ontology. Also they allow users to handle inconsistencies during query formulation through predicate expansion techniques.

Probabilistic graphical models (PGMs). An approach of probabilistic graphical models (PGMs) allows to generate interpretable models that are constructed and then manipulated by reasoning algorithms [71]. These models can also be learned automatically from data, allowing the approach to be used in cases where the manual building of a model is difficult or even impossible.

Statistical Relational Learning (SRL). It is a branch of machine learning that tries to model a joint distribution over relational data [72]. SRL is a combination of statistical learning which addresses uncertainty in data and relational learning which deals with complex relational structures [73].

Inductive logic programming (ILP). Buhmann *et al.* [74] present an approach of inductive lexical learning of class expressions by combining an existing

logical learning approach with statistical relevance measures applied on textual resources.

Pattern extraction. It is an area of work where RDF data is analyzed to extract common patterns, for example, in the form of Frequent Graph Patterns [75] or Statistical Knowledge Patterns [76]. These approaches analyze the underlying RDF data and extract the characteristics related to the ontological axioms based on most frequent patterns. This approach is closely related to the work presented in this thesis.

Our final goal is different from these research approaches. Instead of inducting the ontological axioms, our goal is to induct validation rules. For example, cardinality estimation has been studied in many different domains including relational data. In addition, to integrity constraint validation, it has many other applications such as network monitoring for detecting DDoS attacks or worm propagation, link based spam detection, relation join query optimization. The existing cardinality estimation algorithms such as Hit Counting [77], Adaptive Sampling [78], Probabilistic Counting [79] and HYPERLOGLOG [80] aim to estimate the number of distinct elements in very large set of data with duplicate elements. Neuman and Moerkotte have proposed "characteristic sets" for performing cardinality estimations for RDF queries with multiple joins [81]. These works differ from the work presented in this thesis on two axes. First, they are focused on determining the cardinalities of each value rather than the cardinality of the entity-value relation. Second, they are focused on query optimization rather than integrity constraint validation. However, we use the base of these works such as analysis of mean, variance, and other statistical features to derive an approach for cardinality estimation for integrity constraint validation.

3.4 Summary

In this chapter we discussed the state of the art of quality and validation approaches for KBs. As reflected in this chapter, there is a significant effort in the Semantic Web community to evaluate the quality of a KB. However, in the current state of the art, less focus has been given toward understanding knowledge base resource changes over time to detect anomalies over various releases, which is instead the main

contribution of this thesis. On the other hand, various constraints can be computed in RDF KBs based on their success in the relational model, such as cardinality and range constraints. Ontology based learning and Shape expression languages aim to validate RDF data and to communicate data semantics among users. They cover constraints such as keys and cardinality; however, their expressivity is limited and require user interventions in every step. In this thesis, we aim to automate the process of Shape generation using learning models.

Chapter 4

Evolution Analysis and Quality Characteristics

In this chapter, we explore the concepts of KB evolution analysis and proposed four quality characteristics. We considered the profiling of a knowledge base as a key component of the evolution analysis and the feature extraction process. We investigated the KB evolution analysis using high-level change detection and the factors affecting the evolution. Taking into account RDF dataset profiling and the dynamic features, we proposed four evolution-based quality characteristics. This chapter is structured as follows: Section 4.1 outlines the KB evolution analysis; Section 4.2 discusses the dynamic features from RDF data profiling. Finally, the proposed quality characteristics and measurement functions are illustrated in Section 4.3. This chapter is based on the work presented in [2].

4.1 Evolution Analysis

Large Knowledge Bases (KBs) are often maintained by communities that act as curators to ensure their quality [27]. KBs naturally evolve in time due to several causes: *i)* resource representations and links that are created, updated, and removed; *ii)* the entire graph can change or disappear [82]. We can identify this changes using high-level change detection based on dynamics features. However, the kind of evolution that a KB is subjected to depends on several factors, such as:

- *Frequency of update:* KBs can be updated almost continuously (e.g. daily or weekly) or at long intervals (e.g. yearly);
- *Domain area:* depending on the specific domain, updates can be minor or substantial. For instance, social data is likely to be subject to wide fluctuations than encyclopedic data, which are likely to undergo smaller knowledge increments;
- *Data acquisition:* the process used to acquire the data to be stored in the KB and the characteristics of the sources may influence the evolution; for instance, updates on individual resources cause minor changes when compared to a complete reorganization of a data source's infrastructure such as a change of the domain name;
- *Link between data sources:* when multiple sources are used for building a KB, the alignment and compatibility of such sources affect the overall KB evolution. The differences of KBs have been proved to play a crucial role in various curation tasks such as the synchronization of autonomously developed KB versions, or the visualization of the evolution history of a KB [8] for more user-friendly change management.

Taking into account the above mentioned factors, the benefit of KB evolution analysis can be two-fold [17]: (1) quality control and maintenance; and (2) data exploitation. Considering quality control and maintenance, KB evolution can help to identify common issues such as broken links or URI changes that create inconsistencies in the dataset. Data exploitation can provide valuable insights regarding dynamics of the data, domains, and the communities that explore operational aspects of evolution analysis [17].

4.2 Dynamic Features

Considering data profiling, Ellefi *et al.* [9] presented a set of dynamic features. This set of features explores the changed behavior of KB resources. In particular, it explores the impact of KB resource changes over time. An ideal use case for these features is predicting the availability of resources over various releases. It directly affects the completeness of KB resources. Dynamic features can be used for

measuring the completeness quality characteristics. Based on Ellefi *et al.* [9] study, we explored the following dynamic features:

Degree of change: it helps to understand to what extent the performed update impacts the overall state of the knowledge base. Furthermore, the degree of changes helps to understand what are the causes for change triggers as well as the propagation effects.

Lifespan: knowledge bases contain information about different real-world objects or concepts commonly referred as entities. Lifespan represents the period when a certain entity is available and it measures the change patterns of a knowledge base. Change patterns help to understand the existence and the categories of updates or change behavior.

Update history: it contains basic measurement elements regarding the knowledge base update behavior such as frequency of change. The frequency of change measures the update frequency of KB resources. For example, the instance count of an entity type for various versions.

4.3 Evolution-based Quality Characteristics and Measures

In this section, we define four evolution-based quality characteristics that allow addressing the quality issues (Section 1.1) due to unrestrained evolution. These quality characteristics are fall into two dimensions: intrinsic (those that are independent of user's context), and representational (those that capture aspects related to the design of the data) [22]. Table 4.1 illustrates the quality characteristics with corresponding quality dimensions and dynamic features. In the context of RDF data model our approach focuses on two different types of elements in a KB: classes and properties. At triple level we only explored subjects and predicates thus disregarding the objects either resources or literals. To measure if a certain data quality characteristic is fulfilled for a given KB, each characteristic is formalized and expressed in terms of a measure with a value in the range $[0..1]$.

Table 4.1 Quality characteristics with corresponding quality dimensions and dynamic features.

Dimensions	Characteristics	Features
Intrinsic	Persistency	Degree of change
	Historical Persistency	Lifespan
Representational	Consistency	Update history
	Completeness	

4.3.1 Basic Measure Elements

The foundation of our approach is the change at the statistical level regarding the variation of absolute and relative frequency count of entities between pairs of KB versions.

In particular, we aim to detect variations of two basic statistical measures that can be evaluated with the most simple and computationally inexpensive operation, i.e. counting. The computation is performed on the basis of the classes in a KB (V), i.e. given a class C we consider all the entities t of the type C:

$$\text{count}(C) = |\{s : \exists \langle s, \text{typeof}, C\rangle \in V\}|$$

The count(C) measurement can be performed by means of a basic SPARQL query such as:

```
SELECT COUNT(DISTINCT ?s) AS ?COUNT
WHERE {  ?s a <C> . }
```

The second measure element focuses on the frequency of the properties, within a class C. We define the frequency of a property (in the scope of class C) as:

$$\text{freq}(p,C) = |\{\langle s,p,o\rangle \in V : \exists \langle s, \text{typeof}, C\rangle \in V\}|$$

The freq(p,C) measurement can be performed by means of a simple SPARQL query having the following structure:

```
SELECT COUNT(*) AS ?FREQ
WHERE {
```

```
    ?s <p> ?o.
    ?s a <C>.
}
```

There is an additional basic measurement element that can be used to build derived measures: the number of properties present for the entity type C in the release i of the KB.

$$NP(C) = |\{p : \exists \langle s, p, o \rangle \in V \wedge \langle s, \text{typeof}, C \rangle \in V\}|$$

The $NP(C)$ measure can be collected by means of a SPARQL query having the following structure:

```
SELECT COUNT(DISTINCT ?p) AS ?NP
WHERE {
    ?s ?p ?o.
    ?s a <C>.
}
```

In the remainder, we will use a subscript to indicate the release the measure refers to. The releases are numbered progressively as integers starting from 1 and, by convention, the most recent release is n. So, for instance, $count_{n-1}(\textit{foaf:Person})$ represents the count of resources typed with *foaf:Person* in the last but one release of the knowledge base under consideration. Table 4.2 illustrates two common types of change behaviour using property frequency as measurement element.

Table 4.2 Categories of change behaviour.

Type	**Description**
Stable/Growth = 1	If the property frequency in release N equal or greater than $N-1$
Unstable = 0	If the property frequency in release N less than $N-1$

4.3.2 Persistency

We define the Persistency characteristics as the degree to which erroneous removal of information from current version may impact stability of the resources. Ellefi *et al.* [9] present degree of change feature as an aggregation measure of the dataset dynamics. In this context, Persistency characteristic measure helps to understand stability behaviour of an evolving KB. This quality characteristic provides insights to detect any missing resources in the last KB release.

An additional important feature to be considered when analyzing a knowledge base is that the information stored is expected to grow, either because of new facts appearing in the reality, as time passes by, or due to an extended scope coverage [45]. Persistency measures provide an indication of the adherence of a knowledge base to such continuous growth assumption. Using this quality measure, data curators can identify the classes for which the assumption is not verified. Persistency measure requires atleast two releases of a KB for the measurement function.

The *Persistency* of a class C in a release $i : i > 1$ is defined as:

$$PersistencyClass_i(C) = \begin{cases} 1 & \text{if } count_i(C) \geq count_{i-1}(C) \\ 0 & \text{if } count_i(C) < count_{i-1}(C) \end{cases}$$

the value is 1 if the count of subjects of type C is not decreasing, otherwise it is 0.

Persistency at the knowledge base level – i.e. when all classes are considered – can be computed as the proportion of persistent classes:

$$PersistencyKB_i(KB) = \frac{\sum_{j=1}^{NC_i} PersistencyClass_i(C_j)}{NC_i}$$

where NC_i is the number of classes analyze, and i is the release of the KB.

4.3.3 Historical Persistency

Historical persistency is a derived from the persistency quality characteristic. It captures the whole lifespan of a KB with the goal of detecting quality issues, in

several releases, for a specific entity-type [9]. It considers all entities presented in a KB and provides an overview of the whole KB. It also helps data curators to decide which knowledge base release can be used for future data management tasks. Historical persistency measure requires informations regarding history of KB updates for the measurement function.

The Historical Persistency measure is computed as the average of the pairwise persistency measures for all releases.

$$\text{H\_PersistencyClass}(C) = \frac{\sum_{i=2}^{n} \text{PersistencyClass}_i(C)}{n-1}$$

Similarly to Persistency, it is possible to compute Historical Persistency at the KB level:

$$\text{H\_PersistencyKB}(KB) = \frac{\sum_{i=2}^{n} \text{H\_PersistencyClass}_i}{n-1}$$

4.3.4 Consistency

Consistency characteristics check inconsistent facts that included in a KB due to unrestrained evolution. This quality characteristic relates to the Consistency quality characteristic defined in the ISO/IEC 25012 standard. The standard defines it as the "*degree to which data has attributes that are free from contradictions and are coherent with other data in a specific context of use. It can be either or both among data regarding one entity and across similar data for comparable entities*" [1]. *Zaveri et al.* [22] also explored the Consistency characteristics. In detail, a knowledge base is defined to be consistent if it does not contain conflicting or contradictory facts [22].

We assume that extremely rare predicates are potentially inconsistent, see e.g. the *dbo:Infrastructure/length* property discussed in the example presented in Section 2.6. We can evaluate the consistency of a predicate on the basis of the frequency distribution for an entity type.

We define the consistency of a property p in the scope of a class C:

$$Consistency_i(p,C) = \begin{cases} 1 & if\ \text{freq}_i(p,C) \geq T \\ 0 & if\ \text{freq}_i(p,C) < T \end{cases}$$

Where T is a threshold that can be either a KB-dependent constant or can defined on the basis of the count of the scope class.

Threshold Value Analysis:

We propose a threshold value analysis approach similar to SDValidate approach presented by Paulheim and Bizer[62]. Similarly to the SDValidate approach, we assume that properties with low relative frequency are more error-prone. In this account, in our threshold value analysis, we have explored the frequency distribution of properties to identify the threshold value. Furthermore, instead of one version, we considered multiple versions to assess a threshold value empirically.

We started our threshold value analysis by using a histogram of property frequencies distribution. From our initial observation, it is suitable to say that a good threshold value could be a point where there is a trend present in the distribution. Here the word trend should be interpreted as "the way things are heading", e.g., a possible variation in the property frequency distribution. Concluding this reasoning, we come to the assumption that a good threshold point should be located at an extreme value in the first derivative of our histogram. To identify the changes in the histogram, we simply focus on the kernel density estimation (KDE) [83]. It is a non-parametric way of the density function estimation. Furthermore, the density function is based on univariate probability distribution [84]. In the threshold value analysis, we considered KDE for the following reasons: *(i)* univariate probability distribution is considered due to property frequency is the primary measurement element; *(ii)* frequency distribution of properties is unknown for each KB releases, and *(iii)* update frequency varies with each KB.

In our approach, we use the local minimum of the KDE based on the property frequency distribution as a threshold value. However, in most cases, a priori knowledge must be applied to select the most appropriate threshold [85]. In this account, we chose various trend point such as 50, 100, 200, and 500 to maximize the precision of the qualitative analysis results. On the other hand, the number of properties varies with each KB release. Therefore, we also evaluated the last three releases of a KB to further validate our assumption. From our empirical analysis (Sec. 7.2.3) , we

considered 100 as the threshold value by evaluating properties present in various KB releases that are optimized in the context of our qualitative analysis.

4.3.5 Completeness

ISO/IEC 25012 defines the Completeness quality characteristic as the "*degree to which subject data associated with an entity has values for all expected attributes and related entity instances in a specific context of use*" [1]. In general, completeness consists in the degree to which all required information is present in a particular dataset [22].

Evolution-based completeness focuses on the removal of information as a negative effect of the KB evolution. It is based on the continuous growth assumption as well; as a consequence we expect properties of subjects should not be removed as the KB evolves (e.g. *dbo:Astronaut/TimeInSpace* property described in the example presented in Section 2.6). Completeness measure requires two releases of a KB for the measurement function.

The basic measure we use is the frequency of predicates, in particular, since the variation in the number of subjects can affect the frequency, we introduce a normalized frequency as:

$$\mathrm{NF}_i(p,C) = \frac{\mathrm{freq}_i(p,C)}{\mathrm{count}_i(C)}$$

On the basis of this derived measure we can thus define completeness of a property p in the scope of a class C as:

$$\mathit{Completeness}_i(p,C) = \begin{cases} 1, & \mathrm{NF}_i(p,C) \geq \mathrm{NF}_{i-1}(p,C) \\ 0, & \mathrm{NF}_i(p,C) < \mathrm{NF}_{i-1}(p,C) \end{cases}$$

At the class level the completeness is the proportion of complete predicates and can be computed as:

$$\mathit{Completeness}_i(C) = \frac{\sum_{k=1}^{\mathrm{NP}_i(C)} \mathit{Completeness}_i(p_k,C)}{\mathrm{NP}_i(C)}$$

where $NP_i(C)$ is the number of properties present for class C in the release i of the knowledge base, and p_k.

4.4 Summary

In this chapter, we proposed four quality characteristics using KB evolution analysis that can be used to assess a KB. Further, for each quality characteristics, we provide different measurement functions. Persistency and Historical Persistency measure depends on the degree of change feature of the entity type dynamics. On the other hand, Completeness and Consistency measures use the update history of properties present in a entity type for identifying any issues present in the dataset. These metrics are implemented as part of a tool, namely, KBQ tool [24]. This tool is employed to assess the quality of any KB including our use cases. Detailed analysis and validation of quality characteristics is discussed in Chapter 7.

Chapter 5

RDF Shape Induction

Taking into account KB evolution, we explore the work present by Papavasileiou et al. [8], where they formalize KB evolution based on simple changes at low-level and complex changes at high-level. Authors considered low-level and high-level changes are more schema-specific and dependent on the semantics of data. We adopted a similar approach using instance profiling information as a building block for quality analysis (Chapter 4). In this account, high-level change detection at the instance level, being coarse-grained, cannot capture all possible quality issues. However, they can help to identify common quality issues such as errors in the data extraction and/or in the integration process. Furthermore, in evolving KBs, the higher the level of changes, the more context-dependent the issue becomes, as it is tied with factors such as the domain at hand, the design decision, the underlying data, volume, dataset dynamicity and so on. For example, let us consider the issues of erroneous instance deletion of an entity type. If the schema remains unchanged, high-level changes at the instance level will capture erroneous changes present in an evolving KB. This could help to detect completeness issues for a specific entity type. However, if the version of a KB is deployed with design issues, such as incorrect mappings, a quality analysis using high-level change detection may lead to increasing the number of false positives.

Considering the limitations of high-level change detection and the changes present at the schema level, we have investigated RDF validation using integrity constraints. Traditionally, in databases, constraints are limitations incorporated in the data that are supposed to be satisfied all the time by instances [86]. They are

useful for users to understand data as they represent characteristics that data naturally exhibits [87]. In practical settings, constraints are used for three main tasks: *(i)* specifying properties that data should hold; *(ii)* handle contradictions within the data or with respect to the domain under consideration; or *(iii)* as a help for query optimization. In this thesis, we also explored W3C shapes constraints language (SHACL) [88] for assessing the structural integrity constraints and validation rules for RDF instance data.

There are significant theoretical and practical problems in creating a consistent constraint checking approach using ontologies. For example, the W3C Recommendation OWL, based on Description Logic and the Open World Assumption, was designed for inferring new knowledge rather than for validating data using axioms. Both reasoner and validator have different functions, *i.e.*, a reasoner is used for inferring new knowledge, even though it may find some inconsistencies as well, while a validator is used for finding violations against a set of constraints. In this context, we explore the applicability of a machine learning model to automate the validation process for large KBs.

One of the primary use cases for generating RDF Shapes from data is the quality assessment. In this chapter, we describe the RDF Shape Induction process using SHACL constraints components. In this thesis, we mainly focus on three types of constraints namely, cardinality, range, and string based constraints. We considered these constraints to identify any logical contradiction due to unrestrained KB evolution. Moreover, using these constraints, in this chapter we outline the details of RDF Shape induction process by leveraging on data profiling information. The RDF Shape induction process is based on the work presented in [26]. This chapter is structured as follows: Section 5.1 describes the technical details of RDF shape induction, which is a key step for RDF validation. Furthermore, Section 5.1 outlines the required data profiling information for shape induction process, and the details of RDF Shape induction implementation for feature extraction by analyzing the English DBpedia KB 2016-04 release.

5.1 SHACL Constraints Components and Shape Induction

Consistency checks whether inconsistent facts are included in the KB [22]. For accessing consistency, we can use an inference engine or a reasoner, which supports the expressivity of the underlying knowledge representation formalism. In this context, languages such as W3C Shapes Constraint Language (SHACL) and Shape Expressions Language (ShEx) allow integrity constraints to be defined for RDF data and to validate data. In this thesis, we explored the integrity constraints definitions presented in SHACL core definitions for consistency check. Furthermore, we generate shapes at the class-level using data profiling information. An example excerpt of RDF Shape in SHACL for the *dbo:Person* class is illustrated in Listing 5.1.

Listing 5.1 A snippet from an example Person shape

```
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .

ex:DBpediaPerson a sh:NodeShape;
 sh:targetClass dbo:Person;
 sh:property [sh:path foaf:name;
  sh:minCount 1;
  sh:nodeKind sh:Literal ];
  sh:property [ sh:path dbo:birthDate;
  sh:datatype xsd:date ;
  sh:minCount 1;
  sh:maxCount 1;
  sh:nodeKind sh:Literal ] ;
 sh:property [ sh:path dbo:birthPlace;
  sh:datatype dbo:Place;
  sh:nodeKind sh:BlankNodeOrIRI;
  sh:minCount 1;
  sh:maxCount 1 ] .
```

We consider three constraints for consistency check for evolving KBs: cardinality, range, and string constraint. We consider these three constraints based on the following conditions: *(i)* to evaluate properties with correct specifications, we explore cardinality constraints to identify the correct mapping of properties for a specific

class, and *(ii)* to evaluate contradictions within the data, we explore the range and sting constraints values.

RDF Shapes could help to validate a KB; nevertheless, it is a tedious task to do manually. Thus, to address the challenge of automatically generating RDF Shapes, we propose RDF Shape induction based on data profiling. We define a generic workflow so that it can apply to any constraint. The goal of the workflow is to extract the constraints by analyzing the data patterns and statistics. Using the results from the statistical analysis, we can extract constraints features to generate RDF Shapes that can be used for validation. The main steps of the proposed workflow are illustrated in Figure 5.1.

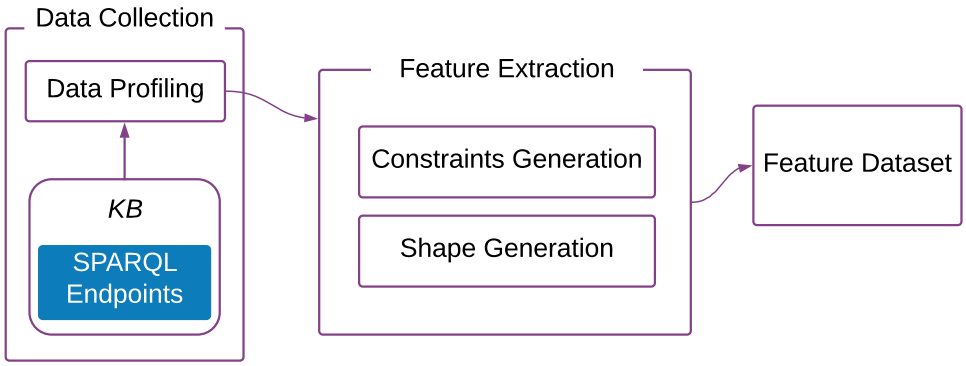


Fig. 5.1 Workflow of profiling based RDF Shape induction.

Our Shape induction workflow contains two main stages: (1) Data Collection, and (2) Feature Extraction. In this context, the data collection phase extracts required statistical informations for constraints generation. Furthermore, the result of the feature extraction phase is the constraints based feature dataset which is then used for predictive modeling and evaluation. More specifically, once the constraint prediction models are built, feature vectors with constraints can be generated.

Taking into account profiling based shape induction tasks, we computed the RDF term at instance-level according to the data instances only. We thereby use the following key statistics: *(i)* Percentage (%) of IRIs, blank nodes, and literals; *(ii)* Triples with IRI and its frequency, length, namespace, patterns; *(iii)* Triples with Literals (String/Numarics/Dates) and its frequency, language, length, patterns, min, max, mean, std, variance.

The motivation for using these key statistics is that these statistics could provide some insights related to different possible distributions to identify feature vectors. The percentage (%) of IRIs, blank nodes, and literals are used to extract Range constraints. Also, statistics of Tiples with IRI is used for Range contraints value extraction. For cardinality and string constraints extraction, we considered the Triples with Literals (String/Numarics/Dates) and its frequency, language, length, patterns, min, max, mean, std, variance. For example, based on the raw cardinality value distributions we can compute the distinct cardinality values. Further, for various distributions we derive 11 statistical measures including min-max cardinalities, mean, mode, standard deviation, variance, quadratic mean, skewness, percentiles, and kurtosis [89]. Our intuition is that these values are descriptive to classify the constraints category. Nevertheless, the data can be noisy, and either min or/and max could be outliers. To address this, we add statistical features that give more insights about the distribution of the cardinalities such as mean, mode, kurtosis, standard deviation, skewness, variance and four percentiles.

In the remainder of this section, we describe these three constraints for shape induction process. We describe each constraint with examples based on the English DBpedia 201604 release.

5.1.1 Cardinality constraints

We observe a trend in vocabularies where the cardinality constraints are explicitly expressed [26]. When we analyzed the 551 vocabularies in the *Linked Open Vocabularies (LOV)* catalogue for the values of owl:minCardinality, 96.91% (848 out of 875) of owl:maxCardinality constraints have value 1 and 93.76% (631 out of 673) of the owl:minCardinality values either 0 or 1 [26]. Thus, in our work rather than trying to estimate the exact values of minimum cardinality and maximum cardinality, we find which cardinality category each property has with respect to a given class. By doing so, we reduce the problem from a regression problem to a classification problem. Table 5.1 shows the common cardinality patterns [26].

In the classification task for cardinality constraints, we identify five main types of cardinality classes: MIN0, MIN1, MIN1+, MAX1, and MAX1+. Out of these, MIN0 and MAX1+ do not put any constraints on the data, such that, any data will be valid for those cardinality types. Thus, if we detect those types, we do not

generate constraints. For other types, corresponding SHACL property constraints are generated as illustrated by Listing 5.2.

Listing 5.2 Cardinality constraints.

```
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .

ex:DBpediaPerson a sh:NodeShape;
 sh:targetClass dbo:Person;
# for MIN1 and MIN1+
 sh:property [sh:path foaf:name;
  sh:minCount 1 ];

# for MAX1
 sh:property [ sh:path dbo:birthDate;
  sh:maxCount 1] .

# for MAX1+
 sh:property [ sh:path dbo:union;
  sh:maxCount 1] .
```

Table 5.1 Minimum and maximum cardinality levels.

Key	Description
MIN0	Minimum Cardinality = 0
MIN1	Minimum Cardinality = 1
MIN1+	Minimum Cardinality >1
MAX1	Maximum Cardinality = 1
MAX1+	Maximum Cardinality >1

We generate cardinality information for each property associated with the instances of a given class. Our goal is to goal is to induct validation rules using data from instance level. The work presented by Neumann and Guido [81] help to identify raw cardinality values using SPARQL queries. They proposed a highly accurate cardinality estimation method for RDF data using star joins SPARQL queries. Similarly, we also explore the process of cardinality values estimation using the results from SPARQL queries. Thus, our cardinality constraints generation process is based

on the study presented by Neumann and Guido [81]. We adopted this approach for cardinality values estimation for each property associated with a given class. More specifically, we collected distinct cardinality values by using star join SPARQL query. An example of join queries for raw cardinality values estimation present in Listing 5.3.

Listing 5.3 SPARQL query for the cardinality value estimation.

```
SELECT ?card (COUNT (?s) as ?count )
WHERE {
  SELECT ?s (COUNT (?o) as ?card)
  WHERE {
     ?s a ?class ;
     ?p ?o
   } GROUP BY ?s
} GROUP BY ?card ORDER BY DESC(?count)
```

5.1.2 Range constraints

We use the subset of the target Node already identified in SHACL, i.e., *IRI*, *Literal*, *BlankNode*, and *BlankNodeOrIRI*. Table 5.2 illustrates the target Node objects type in SHACL. Each value of target Node in shape is either an IRI or a literal. For range constraints, our goals are twofold. First, we want to generate an object as target node constraint for each property associated with a given class. Once the target node type is determined, then more specific range constraints have to be decided. If the node type is *Literal*, the corresponding datatype has to be determined. If the node type is either *IRI*, *BlankNode*, or *BlankNodeOrIRI* the class type of the objects has to be determined.

We classify each property associated with instances of a given class to one of the aforementioned node types. The second task of assigning the corresponding datatype or class as the range of each property is done based on heuristics of datatype or class type distributions among the set of objects associated with the property. For example dbo:Web has distribution of 73.13% for IRI node type and 26.89% for LIT node type for *dbo:SoprtsTeam* entity type. In this account, IRI has larger distribution then LIT node type. Based on our heuristics we considered *dbo:Web* node type as IRI. An example of *dbo:Person-dbp:birthPlace* objects nodeKind constraints Shape is illustrated in the Listing 5.4.

Table 5.2 Objects Type.

IRI	BlankNode	Literal	Type
X	X	X	Any
X	X		BlankNodeOrIRI
X			**IRI**
	X		BlankNode
		X	**Literal**
X		X	IRIOrLiteral
	X	X	BlankNodeOrLiteral

Listing 5.4 Node type constraints.

```
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix dbp: <http://dbpedia.org/property/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .

ex:DBpediaPerson a sh:NodeShape;
 sh:targetClass dbo:Person;
# node type IRI
 sh:property [sh:path dbp:birthPlace;
  sh:nodeKind sh:IRI;
  sh:or ( [sh:class schema:Place]
    [ sh:class dbo:Place ] )
  ];

# node type literal
 sh:property [ sh:path dbp:deathDate;
  sh:nodeKind sh:Literal;
  sh:datatype xsd:date ] .
```

5.1.3 String based constraints

For string based constraints generation the primary focus is to understand minimum length (minLength) and maximum length (maxLength) of a property. In this context

max and min length subjected to rdf:type and node with literal values. In general, if the value of minLength is 0, then there is no restriction on the string length, but the constraint is still violated if the value node is a blank node. On the other hand, the value of maxLength without restriction could be any string length based on the rdf:type. We considered the distribution of string lengths to identify minLength and maxLength of literal values of a property. More specifically, we explored all the properties present in a class, literals string lengths distribution interquartile range for constraints generation. We evaluate the minLength using 1st quartile(Q1) and maxLength using the 3rd quartile (Q3). Table 5.3 illustrates the string length conditions for minLength and maxLength. In particular, we mainly focus on identifying a relative range for the maximum and minimum length. An example of string length based SHACL Shape for *dbo:title* property is presented in Listing 5.5.

Table 5.3 Minimum and maximum String length levels.

Key	**Description**
minLength0	Minimum Length <Q1
minLength1	Minimum Length $\geq$ Q1
maxLength0	Maximum Length <Q3
maxLength1	Maximum Length $\geq$ Q3

Listing 5.5 String constraints.

```
@prefix dbo: <http://dbpedia.org/ontology/>.
@prefix sh: <http://www.w3.org/ns/shacl#>.

ex:DBpediaPerson a sh:NodeShape;
 sh:targetClass dbo:Person;
# minLength
 sh:property [sh:path foaf:name;
  sh:minLength 1;
  sh:maxLength 8];

# for MAX1
 sh:property [ sh:path dbo:birthDate;
  sh:minLength 1;
  sh:maxLength 8] .
```

5.2 Summary

In this chapter, we described an approach for inducing validation rules in the form of RDF shapes by profiling the data and use inductive approaches to extract validation rules. Another use case for inducting shapes consists in describing the data (which is helpful in generating queries or creating dynamic user interfaces). Based on the proposed RDF Shape induction approach, in Chapter 6 we present a validation process in a generic way that applies to any type of constraint using the results from the evolution based quality assessment. Furthermore, Chapter 7 presents the details of a predictive learning evaluation for two types of constraints, namely, cardinality and range type constraints. Although in this chapter we only discussed the shape induction process for three types of constraints, this approach can be extended to other types of constraints, such as value range constraints (min and max values), string constraints (pattern, languagesIn, uniqueLanguage), or property pair constraints (lessThan, lessThanOrEquals, disjoint, equal) [88].

Chapter 6

Evolution-based Quality Assessment and Validation Approach

In the context of quality assessment methodology, the Data Life Cycle (DLC) provides a high-level overview of the stages involved in successful management and preservation of data for any use and reuse process. Moreover, several versions of the data life cycles exist with different attributes considering variations in practices across domains or communities [34]. Data quality life cycle generally includes the identification of quality requirements and relevant metrics, quality assessment, and quality improvement [35, 90]. Debattista *et al.* [35] presents a data quality life cycle that covers the phases from the assessment of data, to cleaning and storing. They show that in the lifecycle quality assessment and improvement of Linked Data is a continuous process. However, we explored the features of quality assessment based on KB evolution. Our reference Data Life Cycle is defined by the international standard ISO 25024 [1]. We extend the reference DLC to integrate a quality assessment phase along with the data collection, data integration, and external data acquisition phase. This phase ensures data quality for the data processing stage. The extended DLC is reported in Figure 6.1.

The first step in building the quality assessment approach was to identify the quality characteristics. Based on the quality characteristics presented in Section 4.3, we proposed a KB quality assessment approach. In particular, our evolution-based quality assessment approach computes statistical distributions of KB elements from different KB releases and detects anomalies based on evolution patterns. The valida-

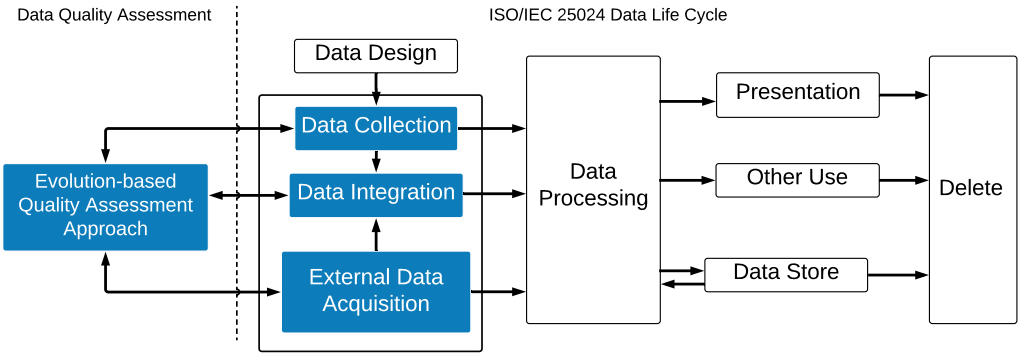


Fig. 6.1 ISO/IEC 25024 Data Life Cycle (DLC) [1] with proposed quality assessment approach. The box highlight the components that are added as improvements to the DLC.

tion approach is based on the RDF Shape induction process introduced in Chapter 5. Figure 6.2 reports the proposed workflow using the quality assessment and validation procedures. In this workflow, the left side displays collection of KB releases as input, while the approach is divided into two phases: (1) *Quality evaluation* (including the statistical profiler) using evolution-based quality characteristics (Chapter 4); and (2) *Validation* which is composed of feature extraction and manual evaluation.

Elaborating further, quality assessment and validation procedures are based on four stages: *(1)* data collection (multiple releases of a knowledge base), *(2)* quality evaluation process (relying on statistical and quality profiling), *(3)* validation process (based on feature extraction and manual validation), and *(4)* modeling and quality problem report (evaluation of learning models and generation of quality problem report). For the quality assessment process, a prototype using the R statistical package is implemented, and it is shared as open source in order to foster reproducibility of the experiments[1].

The rest of this chapter is structured using the proposed quality assessment and validation workflow which is illustrated in Figure 6.2. Section 6.1 provides a general overview of how the multiple releases of a KB are used as input in the approach. The quality evolution process is presented in Section 6.2, where each phase in the pipeline using statistical analysis and quality profiling. The validation process is outlined in Section 6.3 using features extraction (Sec. 6.3.1) process which is based on RDF Shape induction (Chapter 6). Furthermore, Section 6.3.2 presents a detailed

[1]Source: https://github.com/rifat963/KBQ

description of the manual evaluation and the golden standard generation process. Section 6.4 discusses the approach for model evaluation and generation of the quality problem report. Finally, Section 6.5 presents a tool using the proposed quality characteristics and manual validation approach.

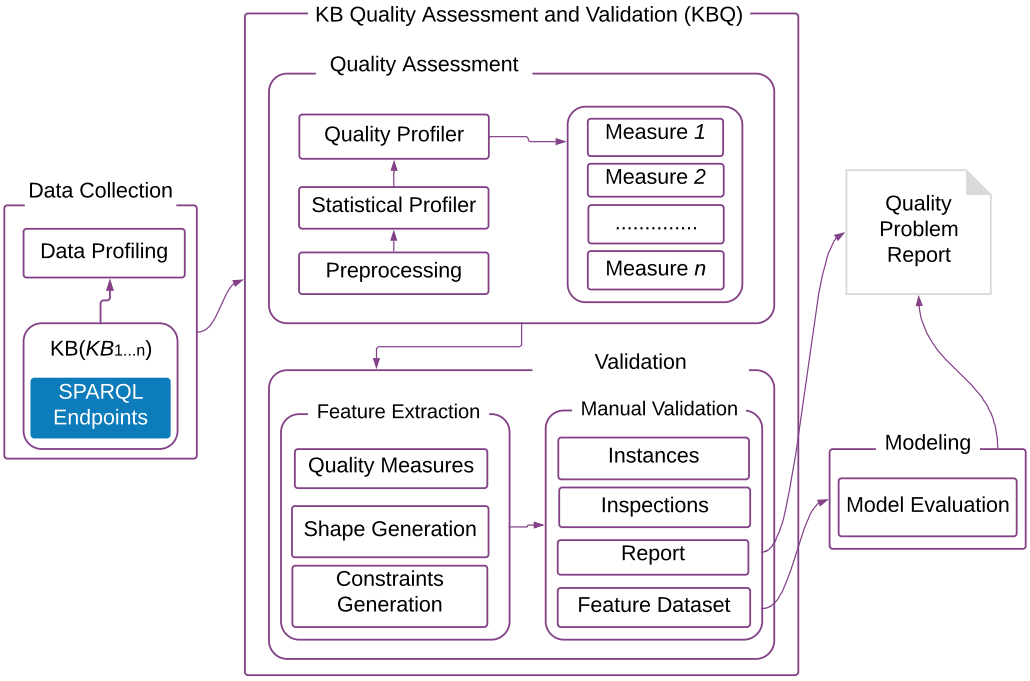


Fig. 6.2 Proposed Quality Assessment and Validation Workflow.

6.1 Data Collection

In this approach, the history of KB releases and summary statistics are applied as inputs to the quality assessment and validation process. Typically the acquisition of KB releases is performed by querying multiple SPARQL endpoints (assuming each release of the KB is accessible through a different endpoint) or by loading data dumps; from the KB releases the summary statistics are generated. Furthermore, the Data Collection component is build on top of Loupe [91], an online system that inspects and extracts automatically statistics about the entities, vocabularies (classes,

and properties), and frequent triple patterns of a KB. An intermediate data structure is created using the entity type and KB releases. This intermediate data structure is used as an input to the next step. Figure 6.3 reports the intermediary data structure that is used in the following stages.

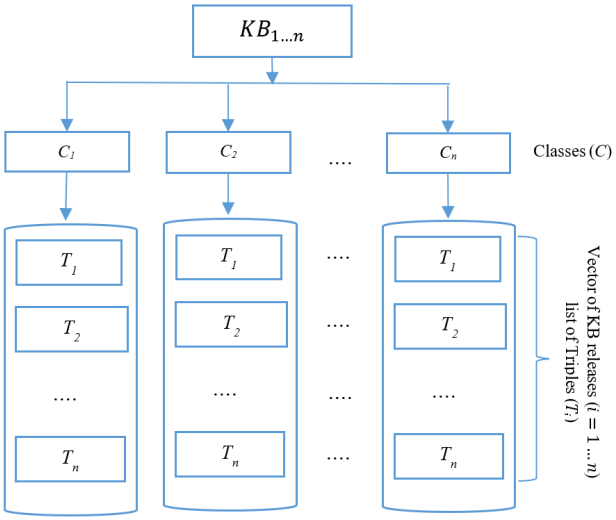


Fig. 6.3 Intermediary data structure that is used as input for the Evaluation Process.

6.2 Quality Evaluation

The proposed quality characteristics are based on the assumption that anomalies can be identified using a combination of data profiling and statistical analysis techniques. The data-driven measurements are based on changes over time in different releases. In this context, the knowledge base quality analysis is performed using quality characteristics presented in Section 4.3. Firstly, the quality characteristics are evaluated by analyzing multiple KB releases; then, the result of quality assessment consists of quality information for each assessed knowledge base. This generates a quality problem report that can be as detailed as pinpointing specific issues at the level of individual triples. These issues can be traced back to data quality problem and can be more easily identified starting from a high-level report. For example, a quality problem report can help to identify incompleteness issues due to unrestrained evolution.

The evaluation process includes the following three steps:

1. **Preprocessing:** In this component, preprocessing operations is performed over the intermediate data structure based on schema consistency checks. In particular, the goal of this component is to check if the chosen entity type is present in all the releases of a KB to verify schema consistency. This is essential to perform the schema consistency checks due to high-level changes that are more schema-specific and dependent on the semantics of data. More specifically, this component does the following tasks: *(i)* selection of only those entity types that are present in all KB releases; *(ii)* and for each entity type, selection of only those predicates present in that class. Furthermore, it is essential to filter those properties for an entity type in the intermediate data structure in case the instance count is 0 for all the KB releases.

2. **Statistical Profiler:** Then, in order to identify the dynamic features of the sequence of KB releases, the following key statistics are computed using basic statistical operations:

   i) number of distinct predicates; *ii)* number of distinct subjects; *iii)* number of distinct entities per class; *iv)* frequency of predicates per entity;

   To identify the KB release changes, it counts the frequency of property values for a specific class. Also, the distinct entity count for a specific class is considered that presented as measurement elements in Section 4.3.1. The change detection between two KB releases is computed by observing the variation of key statistics. The quality characteristics are divided into class and property level. For class level quality characteristics, entity count is considered as the basic measurement elements for change detection. For a particular class, the property level quality characteristics are measured using frequency of properties as essential measurement elements for change detection.

3. **Quality Profiler:** Typically, data profiling is defined as the process of creating descriptive information and collect statistics about the data [57]. It summarizes the dataset without inspecting the raw data. This component use the approach of data profiling together with quality measure to profile quality issues. The statistical profiler is used to analyze the KB releases. For analyzing the KB datasets, it used four quality characteristics presented in Section 4.3. Quality profiler includes descriptive as well as measure values based on the quality characteristics.

Elaborating further, this component does the following tasks: *(i)* it provides statistical information about KB releases and patterns in the dataset (e.g. properties distribution, number of entities and RDF triples); *(ii)* it provides general information about the KB release, such as dataset description of class and properties, release or update dates; *(iii)* it provides quality information about the vector of KB releases, such as quality measure values, list of erroneous analyzed triples.

6.3 Validation Process

The primary goal of the validation process is to analyze the results of the quality evaluation process using manual validation (which is a human-driven task). Furthermore, constraints based features are generated to validate the entities with consistency issues (Chapter 5). Then, the automatically generated features are further validated using manual validation. More specifically, the datasets from the quality evaluation process is considered as input to the validation process. The data validation process is divided into two parts: *(i)* features extraction; and *(ii)* manual validation.

Feature extraction process is based on the results from the quality characteristics (Sec. 4.3) and RDF shape induction (Sec. 5.1). More specifically, using the results from the quality evaluation process, RDF Shape induction process is applied in order to create a constraints-based feature dataset. Furthermore, a qualitative analysis is performed using manual validation to evaluate the precision of quality measures. The RDF Shape induction results are manually evaluated to create a partial gold standard for modeling tasks. The phases of the data validation approach are explained in details in the following sections.

6.3.1 Feature Extraction

To instruct the learning models, a feature extraction task is performed that is composed of *(i)* selecting an entity type using the quality measure results, and *(ii)* constraints based shape induction to compute the features. The features are five in total and they are grouped into three categories based on the quality issues as shown in Table 6.1. More specifically, five features are extracted from quality measures and shape induction process.

i) Persistency features: It is based on the persistency measures (Sec. 4.3). Using the selected entity type from schema consistency check, it evaluate changes present in the current release with respect to the previous release. The result of the persistency features is indicated by a Boolean value of 0 or 1: 1 indicates a normal growth, while 0 indicates an unstable behaviour.

ii) Completeness features: It is based on the completeness measures (Sec. 4.3). Using the selected entity type and properties from schema consistency check, it evaluate the completeness meaures values at the property level. Similar to persistency features, the result of this features is indicated by a Boolean values of 0 or 1: 1 indicates a normal growth, while 0 indicates an unstable behaviour.

ii) Consistency features: It is based on the results from integrity constraint checks that are derived from the SHACL representation (Sec. 5.1). In this analysis, the cardinality and range constraints are applied for consistency evaluation. Moreover, we divide the cardinality constraints into minimum and maximum cardinality constraints. In particular, this phase will validate the persistency feature dataset using three features: *i)* Properties with minimum cardinality values of MIN0 or MIN1+; *ii)* Properties with maximum cardinality values of MAX1 or MAX1+; *iii)* Properties with range values of LIT or IRI. Each of this feature is used as inputs of a binary classifier and applied in a supervised learning fashion to evaluate the constraints datasets.

Table 6.1 Features based on the quality issues.

Quality Issues	**Feature**	**Classifier Values**
Persistency	Entity type	(0,1)
Completeness	Property	(0,1)
Consistency	Minmum Cardinality	(MIN0,MIN1+)
	Maximum Cardinality	(MAX1, MAX1+)
	Range	(IRI,LIT)

6.3.2 Manual Validation and Gold Standard Creation

The main goal of this step is to extract, inspect, and perform manual validation for identifying the causes of quality issues as well as create gold standard. Elaborating further, manual validation tasks are based on the following three steps:

i) Instances: For manual evaluation, a portion of the properties with quality issues is selected based on the quantitative analysis. The selection has been performed in a random fashion to preserve the representativeness of the experimental data. The proposed quality characteristics are based on the results of statistical profiling to identify any missing entities. For manual validation, a set disjoint operation between two releases is performed to identify those missing entities in the last release of a given KB. In particular, in this step, using set disjoint operation all entities are extracted from the last two releases of a given KB.

ii) Inspections: Using the dataset from instance extraction phase, an inspection of each instance is performed for manual validation and report. Various KBs adopt automatic approaches to gather data from the structured or unstructured data sources. For example, DBpedia KB uses an automatic extraction process based on the mapping with Wikipedia pages. For the manual validation, inspection of the sources is performed using the missing instances to identify the causes of quality issues. In particular, manual evaluation check if the information is present in the data sources but missing in the KB.

iii) Report: the validation result of a entity is reported as true positive (the subject presents an issue, and an actual problem was detected) or false positive (the item presents a possible issue, but none actual problem is found).

In this approach, a partial gold standard strategy (Sec. 2.7) is adopted based on the assumption that a new (small) training set is needed when dealing with a new knowledge base. The manual validation phase is then in charge of inspecting and performing a manual annotation of the detected integrity constraints. In detail:

(i) Feature extraction: it selects the entities and properties from the evolution analysis for RDF shape induction. Then, it selects the properties annotated with integrity constraints for further inspection.

(ii) Inspection: the validation result of an instance is reported as *Correct* (the properties are annotated with correct integrity constraint) or *Incorrect* (the item presents a wrong integrity constraint).

(ii) Feature dataset: the outcome of the manual validation tasks is a subset of the feature dataset according to each integrity constraints. This dataset is considered as the training set for the modeling phase.

6.4 Modeling and Quality Problem Report

The goal of the modeling phase is to generate a model and validate the results of the integrity constraints by computing precision, recall, and F-measure (Sec. 2.8). In particular, the main goal of the validation using integrity constraints is twofold: *(i)* creating constraints dataset that can be used for RDF shape induction, and *(ii)* evaluating the performance of the cardinality and range constraints classifier using learning models. The modeling task is run with a *10-fold cross validation* setup and parameter optimization. The performance of constraint classifiers are evaluated using five classical learning models (Section 2.8). These models are selected to evaluate classifiers performance considering the diversity in machine learning algorithms and to identify the best performing model. Based on the empirical analysis the best performing model is applied for the predictive task.

Finally, a quality problem report is generated based on the quality assessment results. The reports contain quality measure computation results as well as summary statistics for each class. The quality problem report provides detailed information about erroneous classes and properties. Also, the quality measurement results can be used for cataloging and preservation of the knowledge base for future data curation tasks. In particular, the Quality Problem Reporting enables, then, a fine-grained description of quality problems found while assessing a knowledge base. The quality problem report visualization is implemented using R markdown documents. R markdown documents are fully reproducible and easy to perform analyses that include graphs and tables. An example of Quality problem report is presented in the GitHub repository[1].

6.5 KBQ : A proof-of-concept

We developed KBQ, a tool[2] for KB quality assessment using evolution analysis. It is based on the quality assessment approach illustrated in Figure 6.2. In particular, we leverage the evolution-based quality characteristics (Chapter 4) as measurement elements for quality issues indicator.

Architecture: KBQ is composed of four modules that are illustrated in Fig. 6.4. We implemented KBQ using the R statistical package that we share as open source in order to foster reproducibility of the experiments[2]. In Appendix A, we outlined the details instruction of KBQ tool in action together with the data extraction REST API. The modules are explained in detail below.

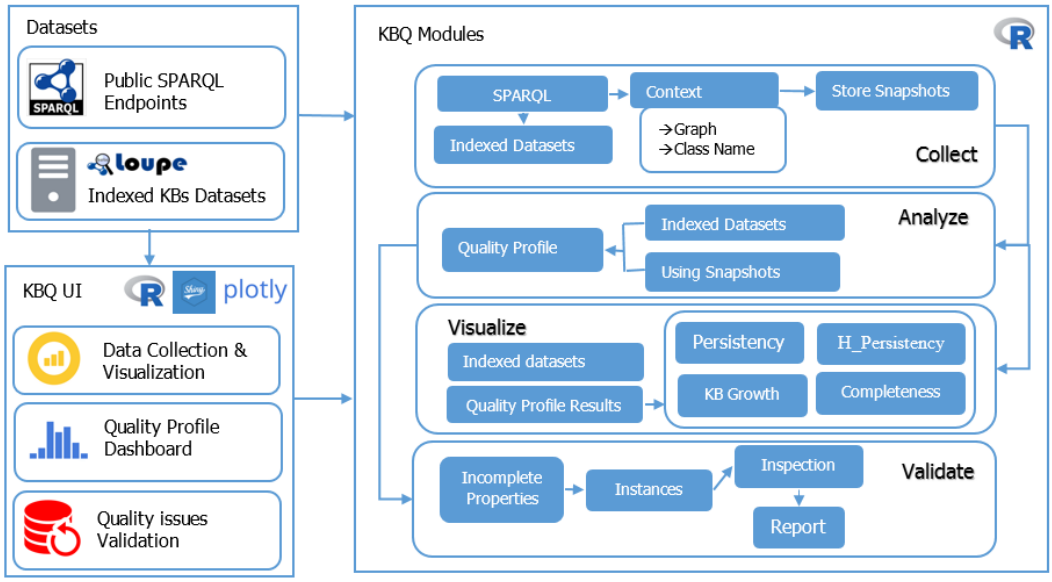


Fig. 6.4 High level architecture of the KBQ tool.

(i) Collect: generates knowledge base (KB) snapshots and sets up timely schedulers. This module is based on the data collection component of the proposed quality evaluation process (Sec. 6.1). It supports (i) collection of KB summary statistics via a dedicated SPARQL endpoint; (ii) collection of periodic KB snapshots that are accessible through a SPARQL endpoint saved in a CSV file. It names each CSV file based on the entity type. Also, it uses a set of APIs[3] for periodic snapshots

[2] Sources:https://github.com/KBQ/KBQ;Website:http://datascience.ismb.it/shiny/KBQ/

[3]The source code is available at https://github.com/rifat963/KBDataObservatory

generation and maintaining scheduled tasks for automatic and timely checks. In particular, it uses the SPARQL endpoint as input and saves the results extracted from the SPARQL endpoints into CSV files.

(ii) Analyze: performs quality profiling based on a particular entity type and generates quality problem report. This module is based on the quality evaluation process outlined in Section 6.2. It uses the intermediate data structure to speed up the execution of the measurement functions. This data structure is created by grouping sets of resources and predicates for an entity type based on KB releases. The evolution-based quality characteristics measures result value of 0 or 1 are used as indicators for the quality issues detection. More specifically, the quality indicators are based on the quality characteristics presented in Section 4.3.

(iii) Visualize: is composed of two modules: (i) list of quality assessment results and (ii) data set catalogue. Visualization of quality assessment results are embedded with analysis module based on four quality characteristics. This allows any user to access quality measures by selecting a specific characteristics. It also allows class faceted exploration along the various KB releases.

(iv) Validate: is based on the manual validation approach introduced in Section 6.3.2. More specifically, validate module composed into two stages: extracts and inspects. Moreover, in KBQ inspects allows manual annotations of quality issues. A user can extract properties with quality issues after performing a quality profiling that consists of: *(i)* Incomplete properties: visualize a list of properties with completeness quality issues for validation; *(ii)* Instances: in the quality evaluation approach profiling is done based on the summary statistics. To extract the missing instances of a property, the instance extraction component performs comparison between the list of instances from the last two versions; *(iii)* Inspections: after the instance extraction is done, a user can select every instance for evaluation and report. It present instance inspection based on data sources. In particular, validation is performed by inspecting the missing instances and manually evaluate cause of quality issues through data source inspections. In this module, each data sources are linked based on the corresponding KBs resource URI; *(iv)* Report: a user can report if the instance is true positive or false positive, as well as a user can comment on specific issues. Finally, a user can save the validation report in a HTML file.

6.6 Summary

In this chapter, we devise a data quality assessment and validation methodology, which is comprised of evolution-based quality characteristics and validation using SHACL constraints components induction. The quality assessment was implemented using KB evolution analysis by monitoring changes presents in each KB release. Furthermore, we introduce a twofold validation approach: (1) RDF data validation using SHACL based integrity constraints induction using predictive modeling; and (2) manual validation to further validate the quantitative results. Moreover, as a proof of our concepts and to perform experimental analysis, we have introduced KBQ-tool based on our quality assessment approach. We reports the experimental results in Chapter 7 and summary of our findings in Chapter 8.

Chapter 7

Experimental Results

This chapter reports an experimental analysis of our approach that has been conducted on two KBs, namely DBpedia and 3cixty Nice. The analysis is based on the quality characteristics and measures described in Section 4.3. The measurement has been conducted by means of a prototype implementation of a tool that is described in Section 6.5. We first present the experimental setting of the implementation in Section 7.1. Then, we report the results of quantitative analysis in Section 7.2. Based on the results from quantitative analysis, we present a qualitative analysis using manual validation in Section 7.3. Finally, Section 7.4 outlines the evaluation results of validation using integrity constraints.

7.1 Experimental Settings

We used the public SPARQL endpoint for each KB and saved the results in the CSV files using KBQ tool. We named each CSV file based on the knowledge base release and corresponding class name. In Figure 7.1, we illustrate the entity type base grouping of extracted CSV files for all DBpedia KB releases. For instance, we extracted all triples of the 11 DBpedia KB releases belonging to the class *foaf:Person* and saved them into CSV files named with the names of the DBpedia releases. Our experimental datasets are presented at [1].

We present a detailed summary of the extracted datasets for each KB.

[1] https://github.com/rifat963/KBQ

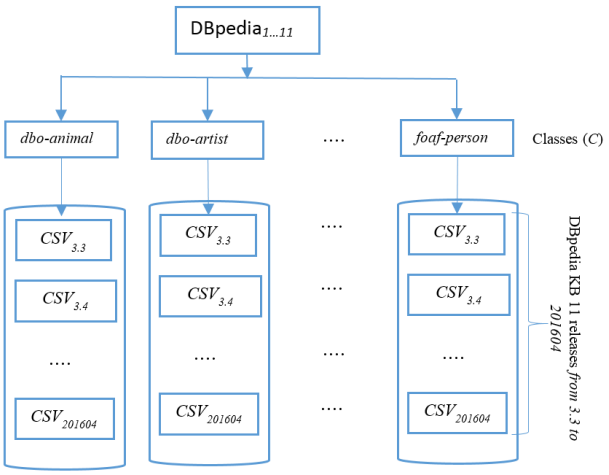


Fig. 7.1 Structure of input module.

3cixty Nice: We used the public SPARQL endpoint of the 3cixty Nice KB in our data extraction module. As the schema in 3cixty KB remains unchanged, we used the same SPARQL endpoint for collecting 8 different releases of 3cixty Nice KB from 2016-03-11 to 2016-09-09. We considered those instances having the *rdf:type* of *lode:Event* and *dul:Place*. These two entity types are the most common according to the total number of entities. The distinct instance count for each class is presented in Table 7.2a. The variation of *count* in the dataset and the observed history is presented in Figure 7.2. From the 3cixty Nice KB, we collected a total of 149 distinct properties for the *lode:Event* typed entities and 192 distinct properties for the *dul:Place* typed entities across eight different releases. To monitor the completeness issues for continuous updates, we collected 50 snapshots of *lode:Event* entity type from 2017-07-27 to 2017-09-16. Table 7.2b reports the entity count of *lode:Event* type using periodic snapshots generation.

DBpedia: We collected a total of 11 DBpedia releases from which we extracted 4477 unique properties. For this analysis we considered the following ten classes: *dbo:Animal*, *dbo:Artist*, *dbo:Athlete*, *dbo:Film*, *dbo:MusicalWork*, *dbo:Organisation*, *dbo:Place*, *dbo:Species*, *dbo:Work*, *foaf:Person*. The above entity types are the most common according to the total number of entities present in all 11 releases. Table 7.1 presents the breakdown of frequency per class. We also explored the Spanish version of DBpedia to further validating

Table 7.1 DBpedia 10 Classes entity count (all classes have *dbo:* prefix except the last one).

Version	Animal	Artist	Athlete	Film	MusicalWork	Organisation	Place	Species	Work	foaf:Person
3.3	51,809	65,109	95,964	40,310	113,329	113,329	31,8017	11,8042	213,231	29,498
3.4	87,543	71,789	113,389	44,706	120,068	120,068	337,551	130,466	229,152	30,860
3.5	96,534	73,721	73,721	49,182	131,040	131,040	413,423	146,082	320,054	48,692
3.6	116,528	83,847	133,156	53,619	138,921	138,921	413,423	168,575	355,100	296,595
3.7	129,027	57,772	150,978	60,194	138,921	110,515	525,786	182,848	262,662	825,566
3.8	145,909	61,073	185,126	71,715	159,071	159,071	512,728	202,848	333,270	1,266,984
3.9	178,289	93,532	313,730	77,794	198,516	178,516	754,415	202,339	409,594	1,555,597
2014	195,176	96,300	336,091	87,285	193,205	193,205	816,837	239,194	425,044	1,650,315
201504	214,106	175,881	335,978	171,272	163,958	163,958	943,799	285,320	588,205	2,137,101
201510	232,019	184,371	434,609	177,989	213,785	213,785	1,122,785	305,378	683,923	1,840,598
201604	227,963	145,879	371,804	146,449	203,392	203,392	925,383	301,715	571,847	2,703,493

Table 7.2 3cixty KB Dataset Summary.

(a) *lode:Event* and *dul:Place* type.

Release	lode:Event	dul:Places
2016-03-11	605	20,692
2016-03-22	605	20,692
2016-04-09	1,301	27,858
2016-05-03	1,301	26,066
2016-05-13	1,409	26,827
2016-05-27	1,883	25,828
2016-06-15	2,182	41,018
2016-09-09	689	44,968

(b) Periodic snapshots of *lode:Event* class.

Release	Entity Count
2017-07-27	114,054
2017-07-28	114,542
2017-07-29	114,544
2017-07-30	114,544
other rows are omitted for brevity	
2017-09-14	188,967
2017-09-15	192,116
2017-09-16	154,745

Table 7.3 Entity Count of Spanish DBpedia KB *dbo:place* class

Release	Entity Count
3.8	321,166
3.9	345,566
2014	365,479
201504	389,240
201510	408,163
201604	659,481
201610	365,479

the causes of quality issues. In Table 7.3, we present the *dbo:place* class entity count across the seven releases of the Spanish DBpedia.

7.2 Quantitative Analysis

We applied our quantitative analysis approach based on the proposed quality characteristics. The goal is to identify any classes and properties affected by quality issues. In particular, we analyzed the aforementioned selected classes from the two KBs to investigate persistency, historical persistency, consistency, and completeness quality characteristics. In Table 7.4, we present the interpretation criteria for each quality characteristic measure. More specifically, in this section, we discuss the quality evaluation results on each use case based on the quality characteristics.

7.2.1 Persistency

3cixty. Table 7.2a reports the entity count measure; in particular we highlight the latest two releases that are considered in computing Persistency according to the definition (Section 4.3). In the case of *lode:Event*-type instances, we can observe that $count_n = 689$ and $count_{n-1} = 2182$, where $n = 8$. Since we have $count_n < count_{n-1}$, the value of *Persistency*(*lode:Event*)-type = 0. That indicate persistency issue present in the last KB release for the *lode:Event* class.

Table 7.4 Verification conditions of the quality measures

Quality Characteristics	**Measure**	**Interpretation**
Persistency	Persistency measure values of 0 or 1	The value of 1 implies no persistency issue present in the class. The value of 0 indicates persistency issues found in the class.
Historical Persistency	Percentage (%) of historical persistency	High % presents an estimation of fewer issues, and lower % entail more issues present in KB releases.
Completeness	List of properties with completeness measures weighted value of 0 or 1	The value of 1 implies no completeness issue present in the property. The value of 0 indicates completeness issues found in the property.
	Percentage (%) of completeness	High % presents an estimation of fewer issues, and lower % entail more issues in KB release.
Consistency	List of properties with consistency measures value of 0 or 1	The value of 1 implies no completeness issue present in the property. The value of 0 indicates completeness issues found in the property.

Similarly, concerning *dul:Place*-type instances, from the dataset we can see that $count_n = 44968$ is greater than $count_{n-1} = 41018$, therefore the value of *Persistency*(*dul:Place*) = 1. Thus, no persistency issue is identified.

We computed 3cixty Nice KB percentage of persistency based on *lode:Events* and *dul:Places* class persistency measure value of 0 or 1. The 3cixty Nice KB percentage of Persistency (%)=$(\frac{\text{No. of classes with issues}}{\text{Total no. of classes}}) * 100 = (\frac{1}{2}) * 10 = 50\%$.

Furthermore, we performed an empirical analysis by monitoring 3cixty KB *lode:Event* entity type. To monitor any changes present for continuous updates, we collected 50 snapshots of *lode:Event* entity type from 2017-07-27 to 2017-09-16. Table 7.2b reports the entity count of *lode:Event* class 50 snapshots which is collected using 3cixty KB SPARQL endpoint. Figure 7.3 illustrates the changes presents in the *lode:Event*-type due to KB growth. There are significant changes present in the last four releases (2017-09-13,2017-09-14,2017-09-15,2017-09-16) entity count. In the 2017-09-13 release, we can see an exponential growth of entity count of 190,1867 compared to previous releases. Furthermore, on the next two releases (2017-09-14,2017-09-15) entity count remains stable due to fewer variation presents in the entity count. However, on the 2017-09-16 snapshots, we can observe a

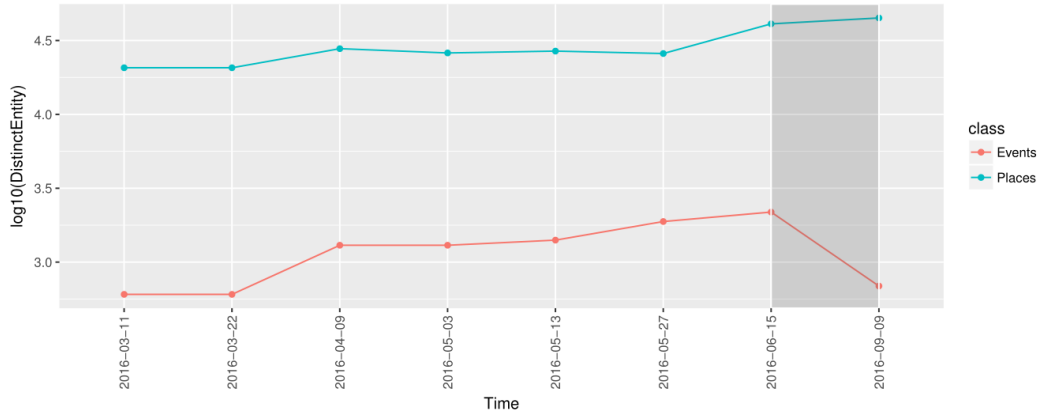


Fig. 7.2 Variation of instances of 3cixty classes *lode:Event* and *dul:Place* over 8 releases.

drop in the entity count which may lead to anomalies in the data integration pipeline. We further investigated the value chain leading to the generation of the KB, and we found an error in the external data acquisition process which leads to missing entities for 2017-09-16 snapshot.

DBpedia. We compare the last two releases (201510, 201604) in terms of entity counts for ten classes; the two releases are highlighted in Table 7.1. The resulting Persistency measure values are reported in Table 7.5. For example, the *foaf:Person* entity counts for the two release (201510, 201604) are respectively $(1,840,598 < 2,703,493)$, thus we find no persistency issue. However, Persistency for the remaining nine classes is 0 since the entity counts in version 201604 are consistently lower than in version 201510. This implies that when DBpedia was updated from version 201510 to 201604, Persistency issues appeared in the DBpedia for nine classes, the exception being only *foaf:Person*. Furthermore, for the Spanish version of the DBpedia *dbo:Place* class 201610 release entity count value of 365,479 is less than 201604 release entity count value of 659,481. Similarly. this implies that persistency issues may arise for the *dbo:Place* class release of 201610.

Discussion. According to the interpretation criteria reported in Table 7.4 we summarize our findings:

- In the case of 3cixty Nice KB, *lode:Event* class, $Persistency = 0$. More in detail, if we consider the two latest releases (i.e. 2016-06-15, 2016-09-09) of the KB and we filter by the type *lode:Event*, the distinct entity counts are equal to 2182 and 689 respectively. Apparently more than 1400 events disappeared

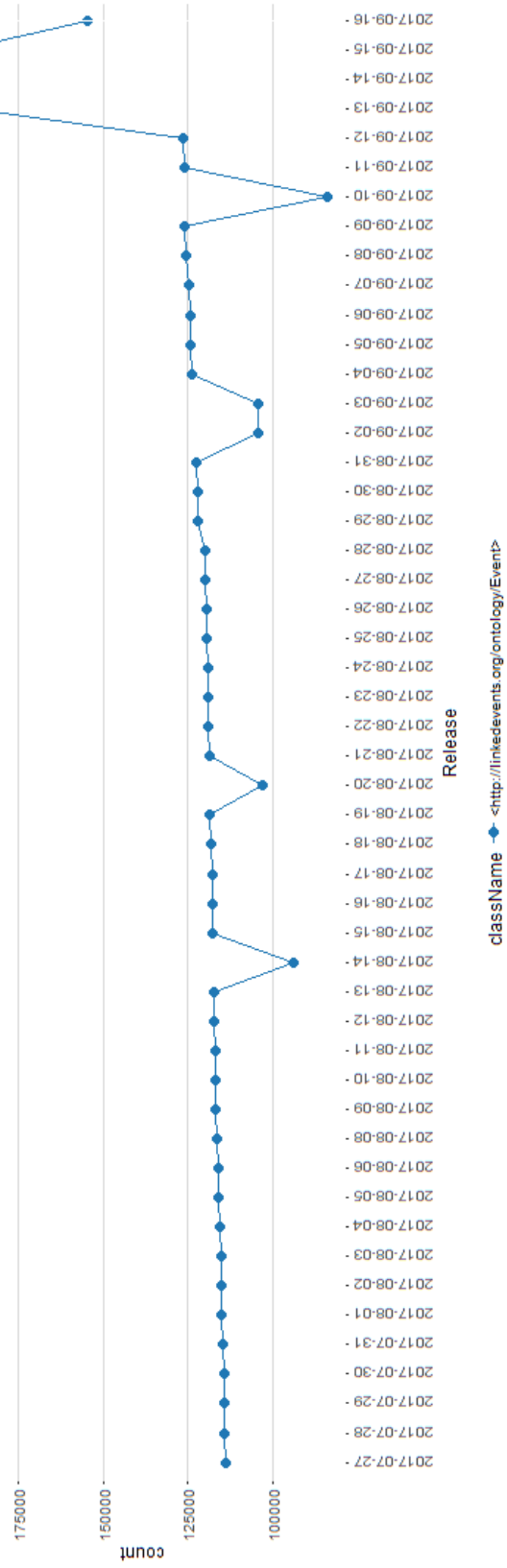


Fig. 7.3 3cixty KB lode:Event class 50 snapshots of entity count.

Table 7.5 DBpedia Persistency and Historical Persistency

Class	Persistency latest release	Releases with Persistency = 0	Historical Persistency
dbo:Animal	0	[201604]	89%
dbo:Artist	0	[3.7, 201604]	78%
dbo:Athlete	0	[201504, 3.5, 201604]	67%
dbo:Film	0	[201604]	89%
dbo:MusicalWork	0	[3.7, 2014, 201504, 201604]	56%
dbo:Organisation	0	[2014, 201604]	78%
dbo:Place	0	[201604]	89%
dbo:Species	0	[201604]	89%
dbo:Work	0	[3.7, 201604]	78%
foaf:Person	1	[201510]	89%

in the 2016-09-09 release: this indicates a potential error in the 3cixty Nice KB. For both investigated types, the percentage of knowledge base Persistency is 50%, which triggers a warning concerning a potential persistency issue existing in the latest (2016-09-09) KB release.

- In the case of DBpedia KB, the analysis conducted using the persistency measure, only the *foaf:Person* class has persistency measure value of 1 indicating no issue. Conversely, all the remaining nine classes show persistency issues as indicated by a measure value of 0. The DBpedia version with the highest number of inconsistent classes is 201604, with a percentage of persistency is equal to 10%.

- The Persistency measure is an observational measure. It only provides an overview of the KB degree of changes. It is effective in the case of rapid changes such as *lode:Event* class.

7.2.2 Historical Persistency

3cixty The variations of persistency measure are considered between the 2016-06-15 and the 2016-09-09 releases. The computation starts from the persistency measures presented in Table 7.2a. For *lode:Event*-type entities the number of persistency variations (with value of 1) $= 6$. Therefore, concerning the *lode:Event*-type the percentage of historical persistency measure value= $(\frac{6}{7}) * 100 = 85.71\%$.

Similarly, for *dul:Place*, the number of persistency variation with value of 1 present over 8 releases $= 5$. In particular, persistency measure value of 0 presented among four releases, (2016-04-09, 2016-05-3) and (2016-5-13, 2016-05-27). So, for the *dul:Place*-type the historical persistency measure assumes the value= $(\frac{5}{7}) * 100 = 71.42\%$.

DBpedia. Figure 7.4 reports the evolution of the 10 classes over the 11 DBpedia releases investigated in our analysis; the diagram highlights the area corresponding to the latest two versions (201510, 201604). The measurement values are reported in Table 7.5 in the rightmost column. The results of *dbo:Animal*, *dbo:Film*, *dbo:Place* and *foaf:Person* classes show only one persistency drop over all the releases. However, *dbo:MusicalWork* has four persistency value of 0 over all releases. The *dbo:MusicalWork* class has the highest number of variations over the release which leads to a low historical persistency value of $(\frac{5}{9}) * 100 = 55.55\%$. Similarly, for the Spanish version of the DBpedia *dbo:Place* class we found entity count variation on the last release (201610). So, for the Spanish DBpedia *dbo:Place*-type the historical persistency measure assumes the value of 89%.

Discussion. The Historical Persistency quality measure provides an overview of the different KB releases. It identifies those versions with persistency issues along the different KB releases. To recap:

- In the case of the 3cixty Nice KB, the *lode:Event* class has one drop (2016-06-15, 2016-09-09) and *dul:Place* class has two (2016-04-09, 2016-05-3), (2016-5-13, 2016-05-27). Thus, overall historical persistency measure of *lode:Event* class higher than *dul:Place* class.
- In the case of DBpedia KB, looking at the Historical Persistency results, *foaf:Person* has persistency value of 0 over the releases of 201504 and 201510. Such values may represent a warning to any data curator interested in the past

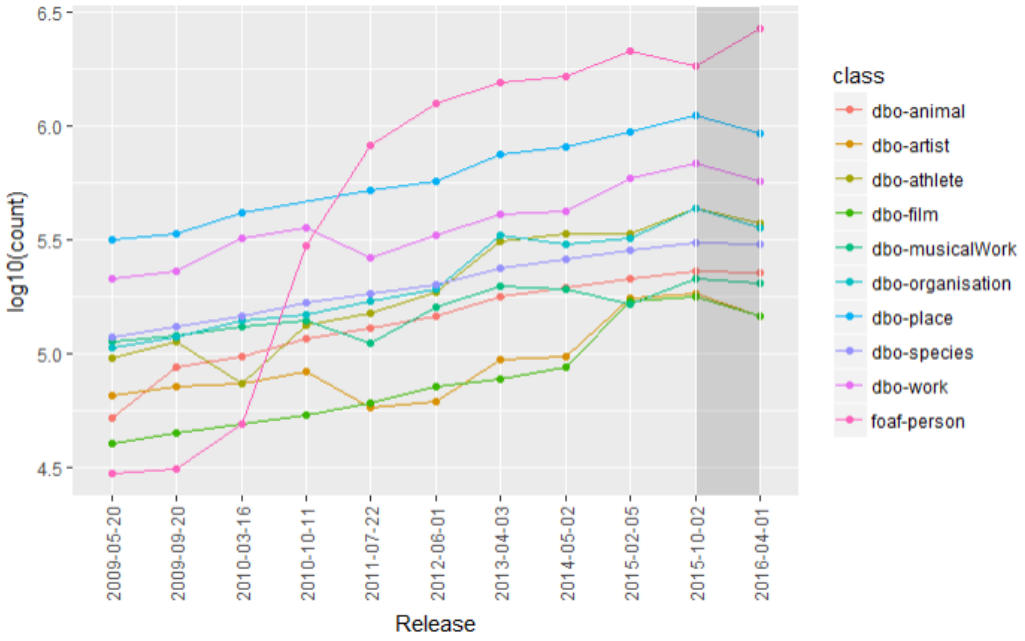


Fig. 7.4 DBpedia 10 Classes instance variation over 11 releases.

evolution of the KB. From the release 3.3 to 201604, *dbo:MusicalWork* shows the lowest values of persistency as a result the historical persistence is 55.55%.

- Historical Persistency is mainly an observational measure and it gives insights on lifespan of a KB. Using this measure value, a data curators can study the behaviour of the KB over the different releases. An ideal example is represented by the *foaf:Person* class. From the results, we observe that for the last two releases (201510, 201604) *foaf:Person* is the only class without persistency issues.

7.2.3 Consistency

Threshold Value. In this experimental analysis, we started by observing histogram of property frequencies distribution and kernel density estimation. For example, 3cixty Nice *lode:Event*-type releases (2016-05-27, 2016-06-15, 2016-06-09) frequency value of 150 has (12, 16, 10) properties, 100 has (12, 15, 10) properties and 50 has (2, 3, 2) properties. In this use case, we found a small number of properties with infrequent distribution. On the other hand, DBpedia KB *foaf:Person*-type frequency distribution for three releases (201504,201510,201604) with the threshold value

of 200 has (178,177,167) properties, 100 has (164,164,158) properties, and 50 has (154,134,126). Figure 7.5 illustrates DBpedia *foaf:Person* class property frequencies distribution. From the *foaf:Person* class kernel density estimation based on three releases, the average value of local minimum is 87.63. In this account, the threshold value of 50 is lower than the local minimum and has the lowest number of properties. On the other hand, the threshold value of 100 is near to the local minimum. Also, the threshold value of 100 has the maximum number of properties which is optimized for our qualitative analysis approach. Thus, we chose 100 since from the empirical analysis at property level it allowed to maximize the precision of the approach.

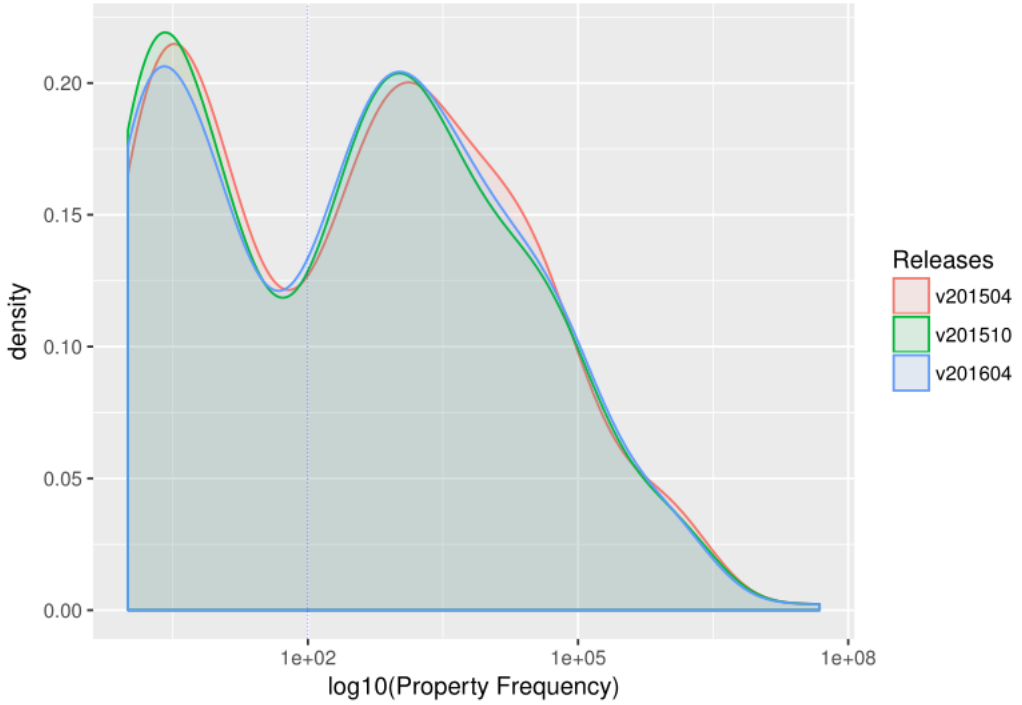


Fig. 7.5 DBpedia *foaf:Person* class property frequencies distribution.

3cixty. we focus on the latest release (2016-09-09) of the 3cixty Nice KB. We analyzed *lode:Event*-type and *dul:Place*-type instances. Based on the threshold values of 100, we measured the consistency for *lode:Event* and *dul:Place*-type. From the *lode:Event*-type resources, by applying the consistency analysis, we found that 10 properties reported below the threshold. Similarly, for *dul:Place*-type resources we found that 12 properties below the threshold value.

DBpedia. Table 7.6 reports, for the DBpedia ten classes, the total number of properties, the *inconsistent* properties – i.e. those with consistency value = 0 –, and

the consistent properties – consistency value = 1. The values are based on the last release 201604. We measured the consistency by identifying those properties with the frequency lower than the threshold value $T = 100$. For example, *foaf:Person* has a total of 381 properties in the 201604 release. We found 158 inconsistent properties, i.e. properties whose frequency is lower than the threshold.

Table 7.6 Properties for the DBpedia classes and Consistency measures. Results are based on Version 201604 with threshold T=100.

Class	Total	Inconsistent	Consistent
dbo:Animal	162	123	39
dbo:Artist	429	329	100
dbo:Athlete	436	298	138
dbo:Film	450	298	152
dbo:MusicalWork	325	280	45
dbo:Organisation	1014	644	370
dbo:Place	1,090	589	501
dbo:Species	99	57	42
dbo:Work	935	659	276
foaf:Person	381	158	223

Discussion. The consistency measure is based on the assumption that properties with low relative frequency more error-prone and applicable to all KB releases. More specifically, we are interested in identifying properties with low relative frequency for an entity type. The main findings are:

- The consistency measure identifies only those properties whose frequency is below the threshold value, which triggers a warning to a data curator concerning a potential consistency issue exist.
- In the 3cixty Nice KB latest release (2016-09-09), we only found ten properties for *lode:Event*-type and twelve for *dul:Place*-type resources. We further investigate this output in the qualitative analysis.
- In the last release (201604) of the DBpedia KB, we have identified consistent properties for 10 classes. Consistency measure results illustrated in Table 7.6.

For example *foaf:Person* class has 158 inconsistent properties. We further investigate this measure for *foaf:Person* class through manual evaluation.

7.2.4 Completeness

3cixty. The measure has been computed based on the last two KB releases, namely 2016-05-15 and 2016-09-09. In Table 7.7, we present a subset of completeness measure results. For the *lode:Event* entity type, the number of predicates in the last two releases = 21 and the number of predicates with completeness issues (value of 0) = 8. In Figure 7.6, we report the measure of completeness for the *lode:events*-type where we only present those properties with issues (value of 0). The percentage of completeness for *lode:Event*-type is $(\frac{13}{21}) * 100 = 62\%$.

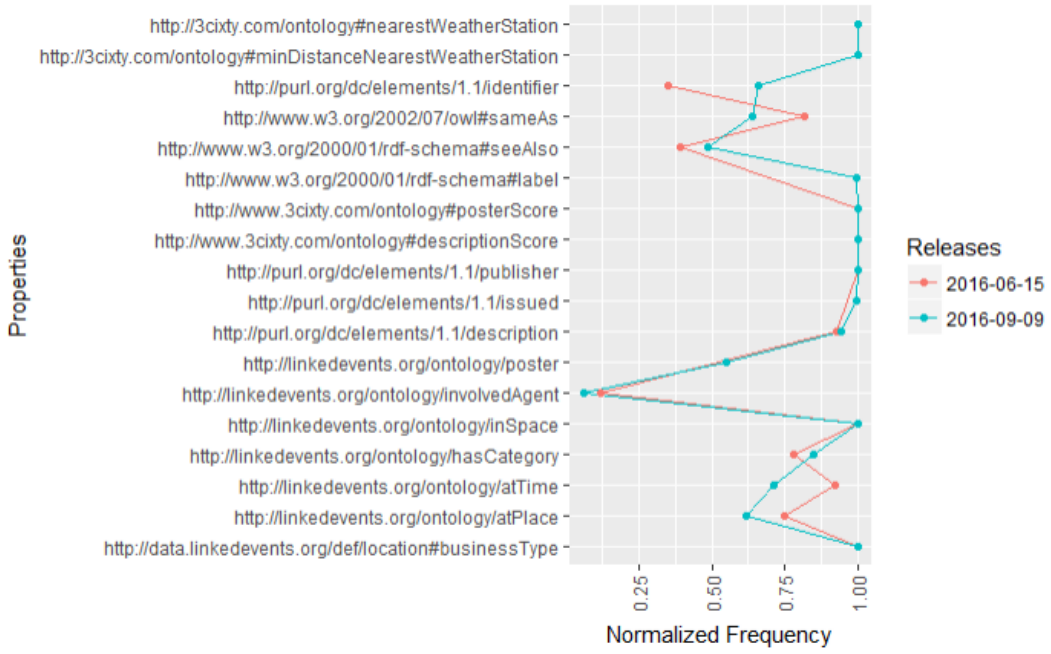


Fig. 7.6 3cixty *lode:Event* completeness measure results

In Table 7.8, we present completeness measures based on 50 periodic snapshots. For example, based on the frequency count of *businessType* in the 2017-09-15 snapshot the observed value is $(1,74,421)$ lower than 2017-09-16 snapshots value $(99,996)$. In this account, the completeness measure value is 0 leading to possible quality issues.

Table 7.7 Completeness measure of 3cixty Nice *lode:Event* class.

Property	2016-05-15	2016-09-09	Complete
atPlace	1,632	424	0
atTime	2,014	490	0
businessType	2,182	689	0
hasCategory	1,698	584	1
other rows are omitted for brevity			
involvedAgent	266	42	0

Table 7.8 Completeness measure of 3cixty Nice *lode:Event* class properties from periodic snapshots.

Property	2017-09-15	2017-09-16	Complete
minDistanceNearestWeatherStation	2,067	2,063	0
nearestWeatherStation	2067	2063	0
businessType	1,74,421	99,996	0
minDistanceNearestMetroStation	72,606	72,606	1
other rows are omitted for brevity			
created	118,070	43,861	0

Similarly, for *dul:Place*-type, the number of predicates in the last two releases $= 28$ and the number of predicates with completeness issue (value of 0) $= 14$. Table 7.9, report a subset of completeness measure results. Figure 7.7, we present *dul:Place*-type completeness measure results of those properties with completeness issue (value of 0). The percentage of completeness for the *dul:Place*-type is equal to $(1 - \frac{14}{28}) * 100 = 50\%$.

DBpedia. Table 7.10 illustrates the results of the completeness measure based on the latest two releases of DBpedia 201510 and 201604. This table reports the completeness measure, for each class, the total number of properties, the complete properties, the incomplete properties, and the percentage of complete properties.

For example, Table 7.11 reports the results of completeness based on the latest two releases of DBpedia 201510 and 201604 for *foaf:Person* entity type. *foaf:Person* has a total of 396 properties over the two considered versions. We computed the

Table 7.9 Completeness measure of 3cixty Nice *dul:Place* class.

Property	**2016-05-15**	**2016-09-09**	**Complete**
poster	37570	44968	1
description	435	600	1
priceRange	67	60	0
aggregateRating	1720	1280	0
other rows are omitted for brevity			
importance	66	67	1

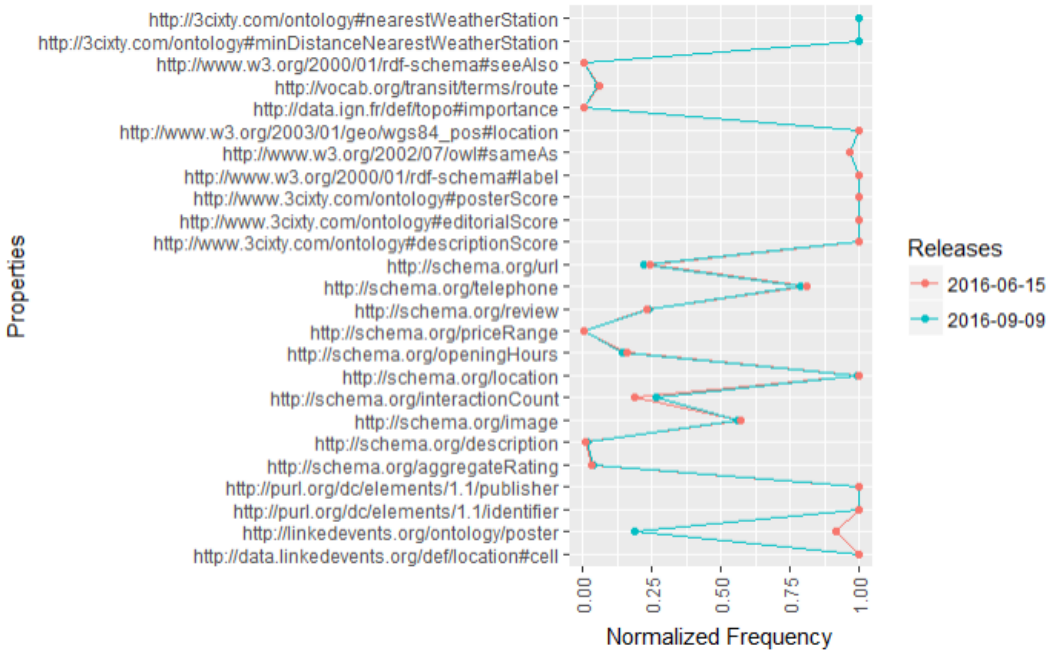


Fig. 7.7 3cixty *dul:Place* completeness measure results

completeness measures over those 396 properties and identified 131 properties with completeness measure value of 0 (incomplete). The remaining 265 properties can be considered as complete. The percentage of complete properties can be computed as $(\frac{265}{396}) * 100 = 66.92\%$.

Taking into account the Spanish version of the DBpedia on the last two releases (201604, 201610) there are in total 8659 common properties present in the datasets. We identified 3606 properties with quality issues based on the frequency difference between two releases. In Table 7.12, we present a subset of completeness measure

Table 7.10 DBpedia 10 class Completeness measure results based on release 201510 and 201604.

Class	Properties	Incomplete	Complete	Complete(%)
dbo:Animal	170	50	120	70.58%
dbo:Artist	372	21	351	94.35%
dbo:Athlete	404	64	340	84.16%
dbo:Film	461	34	427	92.62%
dbo:MusicalWork	335	46	289	86.17%
dbo:Organisation	975	134	841	86.26%
dbo:Place	1,060	141	920	86.69%
dbo:Species	101	27	74	73.27%
dbo:Work	896	89	807	90.06%
foaf:Person	396	131	265	66.92%

results. We have detected quality issues based on the property frequency difference between the two versions for the *dbo:Place* class. For example, the property *dbo:anthem* count is 316 for the 201610 release while it was 557 in the 201604 release. This implies 241 resources missing in the 201610 version of the DBpedia 201610 release.

Table 7.11 Completeness measure of DBpedia KB *foaf:Person* class.

Property	201510	201604	Complete
dbo:timeInSPace	465	419	0
dbo:height	139,445	148,192	1
dbo:weight	67,412	66,144	0
dbo:abstract	1,282,025	1,165,251	0
other rows are omitted for brevity			
dbo:activeYearsEndDate	26,483	25,221	0
dbo:firstRace	796	788	0

Table 7.12 Spanish DBpedia *dbo:Place* class completeness measure based on release 201604 and 201610.

Property	**201604**	**201610**	**Complete**
dbo:abstract	363,572	655,233	1
dbo:address	17,636	13,3781	0
dbo:anthem	557	316	0
dbo:archipelago	3,162	1,871	0
dbo:architect	4,580	2,291	0
dbo:architecturalStyle	6,919	4,373	0
other rows are omitted for brevity			
dbo:area	6,764	3,619	0

Discussion. In general, the completeness measure is based on a pairwise comparison of releases. In this experimental analysis, we compared the last two releases to identify missing data instances in the last release. Below we summarize our findings:

- Looking at the two latest releases (2016-06-15, 2016-09-09) of the 3cixty Nice KB, we have identified those properties with completeness value of 0 as issue indicator. The total number of properties of the latest two versions are 21 excluding those properties not presented in both releases. For instance, the *lode:Event* class property *lode:atPlace*[2] exhibits an observed frequency of 1632 in release 2016-06-15, while it is 424 in release 2016-09-09. As a consequence the Completeness measure evaluates to 0, thus it indicates an issue of completeness in the KB. In 3cixty, the *dul:Place*-type percentage of completeness is 50%, such a figure indicates a high number of incomplete predicates in the latest version (2016-09-09).

- For DBpedia KB looking at the last two releases (201510,201604) we identified incomplete properties for 10 classes. Completeness measure results are listed in Table 7.10. For instance, we identified a total of 131 incomplete properties for *foaf:Person* class. The *foaf:Person* class property *dbo:firstRace* exhibits an observed frequency of 796 in release 201510, while it is 788 in release 201604. As a consequence the completeness measure evaluated to 0, thus it

[2] http://linkedevents.org/ontology/atPlace

indicates an issue of completeness in the KB. We further validate our results through manual inspection. In DBpedia, the *(*foaf:Person) class percentage of completeness is 66.92%, such figure indicates a high number of incomplete instances in the last release (201604).

7.3 Qualitative Analysis using Manual Validation

The general goal of our quality assessment approach is to verify how the *evolution analysis of the changes observed in a set of KB releases helps in quality issue detection*. In the quantitative analysis, we identified classes and properties with quality issues. We, then, summarize on the qualitative analysis based on the results of the quantitative analysis.

Given the large number of resources and properties, we considered just a few classes and a portion of the entities and properties belonging to those classes in order to keep the amount of manual work to a feasible level. The selection has been performed in a total random fashion to preserve the representativeness of the experimental data. In particular we considered a random subset of entities. In general, a quality issue can identify a potential error in the KB. In this account, we focused on the effectiveness of the quality measures when it is able to detect an actual problem in the KB. Manual validation is performed based on the steps introduced in Section 6.3.2

In particular, using the interpretation criteria reported in Table 7.4, from the measure value we can identify a quality issue. The results are a set of potential problems, part of them are accurate – they point to actual problems –, while others are not – they point to false problems. We decided to measure the precision for evaluating the effectiveness of our approach. Precision is defined as the proportion of accurate results of a quality measure over the total results. More in detail, for a given quality measure, we define an item – either a class or a property – as true positive (*TP*) if, according to the interpretation criteria, the item presents an issue and an actual problem was detected in the KB. An item represents a false positive (*FP*) if the interpretation identifies a possible issue but none actual problem is found. The precision can be computed as follows:

$$p = \frac{TP}{TP+FP}. \tag{7.1}$$

We evaluated the precision manually by inspecting the results marked as issues from the completeness and consistency measures.

We considered the results obtained by the quantitative analysis for the entities types and properties attached to the class *lode:Event* for the 3cixty Nice KB; we considered entities and properties related to the classes *dbo:Place*, *dbo:Species*, *dbo:Film* and *foaf:Person* for the DBpedia KB. We designed a set of experiments to measure the precision as well as to verify quality characteristics. In Table 7.13, we present an overview of our selected classes and properties along with the experiments and, in Table 7.14, we summarize the manual evaluation results.

Table 7.13 Selected classes and properties for manual evaluation.

KB	**Level**	**Experiment**
3cixty Nice	Class	Event class to verify Persistency and Historical Persistency.
	Property	*lode:Event* 8 properties from completeness measure to verify and compute precision for completeness.
	Property	*lode:Event* 10 properties from consistency measure to verify and compute precision for consistency.
DBpedia	Class	*dbo:Species* and *dbo:Film* class to verify persistency and historical persistency
	Property	*foaf:Person* and *dbo:Place* class 50 properties from completeness measure to verify as well as compute precision.
	Property	*foaf:Person* class 158 properties and *dbo:Place* class a subset of 114 properties from consistency measure to verify as well as compute precision.

Table 7.14 Summary of manual validation results

Characteristics	3cixty Nice	DBpedia	Causes of quality issues
Persistency & Historical Persistency	True positive: *lode:Event* entities missing due to algorithm error	False positive; *dbo:Species* and *dbo:Film* class quality issues fixed in current version;	*3cixty:* instances are missing due to an error in the reconciliation framework. *DBpedia:* erroneous schema presented in 201510 version has been fixed in 201604 version resulting in a false positive outcome.
Consistency	False Positive: *lode:Event* properties 2016-09-09 release we did not find any error	True positive ; *foaf:Person* and *dbo:Place* class on 201604 version we identify properties with consistency issue. Based on the threshold value of 100 it has a precision of 68% and 76%.	*3cixty:* In this use case, the schema remains consistent for all the KB releases and no real issues were found in the properties with low frequencies. *DBpedia:* In this use case, the schema evolves with each release. We found issues in the properties due to erroneous conceptualization.
Completeness	True positive: *lode:Event* properties missing due to algorithm error. Over 8 properties we computed Precision of 95%	True positive: *foaf:Person* properties missing due to completeness issue. Over 50 properties we computed Precision of 94%. For *dbo:Place* over 50 properties we computed precision of 86%.	We found completeness issues due to data source extraction error for both *3cixty* KB and *DBpedia* KB.

7.3.1 Persistency & Historical Persistency

For persistency and historical persistency, we have investigated a subset of resources for an entity type. The primary motivation is to detect the causes of quality issues for that entity type. Historical persistency is derived from persistency characteristic. We argue that by persistency measure validation, we also verified historical persistency results. Therefore we only performed the validation for persistency.

We evaluated the persistency measure based on the number of entity counts for *lode:Event*-type between two different KB releases (2016-06-15, 2016-09-09) of the 3cixty Nice KB. From the quantitative analysis, we detected *lode:Event*-type has persistency issue with measure value of 0.

For what concerns DBpedia, out of the ten classes under investigation, nine of them have persistency value of 0, which implies that they have persistency issue. We investigated *dbo:Species* and *dbo:Film* entity type that shows issues.

3cixty. From the extracted KB release on 2016-06-15, there are 2,182 distinct entities of type *lode:Event*. However, in the 2016-09-09 release, that figure falls down to 689 distinct entities. We perform a comparison between the two releases to identify the missing entities. As a result we identified a total of 1911 entities missing in the newest release: this is an actual error. After a further investigation with the curators of the KB we found that this is due to an error in the reconciliation framework caused by a problem of overfitting. The error present in the 2016-09-09 release is a true positive identified by the Persistency measure.

DBpedia. We analyzed entity counts of class *dbo:Species* for the latest two releases of DBpedia (201510 and 201604). The counts are 305,378 and 301,715 respectively. We performed a comparison between the two releases to identify the missing entities; we found 12,791 entities that are no more present in the latest release. We investigate in detail the first six missing entities. For example, the entity *AIDS_II* [3] in 201510 was classified with type *dbo:Article* as well as *dbo:Species*. However, in 201604 it has been updated with a new type and the type *dbo:Species* was removed. There was clearly an error in the previous version that has been fixed in the latest, however, from the point of view of the latest release this is a false positive.

We performed fine grained analysis based on subset of entity type *dbo:Film* for the latest two releases of DBpedia (201510 and 201604). The counts are 177,989 and 146,449 respectively. We performed a comparison between the two releases to identify the missing entities; we found 49,112 entities that are no more present in the latest release. We investigate in more detail the first six missing entities. For example, the subject *dbpedia:$9.99*in 201510 was classified with type *dbo:Work* as well as *dbo:Film*. However, in 201604 it has been removed from both *dbo:Work* and *dbo:Film* was removed. We further explore the Wikipedia page *wikidpedia-en:$9.99* and film exists. It is clearly an error in the data extraction in 201604 release.

[3] http://dbpedia.org/page/AIDS_(computer_virus)

7.3.2 Consistency

We computed the consistency measure values using the threshold $T = 100$. Properties with consistency = 0 were considered as potential quality issues. We considered the properties attached to entities typed *lode:Event* for the 2016-09-09 3cixty Nice KB. For the DBpedia KB, we considered the properties attached to the entities of type *foaf:Person* and *dbo:Place* from the 201604 release.

3cixty. We found only 10 inconsistent properties. After a manual inspection of those properties we were unable to identify any actual error in the resources, so we classified all of the issues as false positives. In 3cixty KB schema remains consistent for all the releases. We identified that these properties common for all instances and we didn't find any erroneous conceptualization in the schema presentation.

DBpedia. We extracted all the properties attached to entities of type *foaf:Person* and we identified 158 inconsistent properties. From the properties list, we inspected each of the property resources in detail. From the initial inspection, we observe that properties with low frequency contain actual consistency problems. For example, the property *dbo:Lake* present in the class *foaf:Person* has a property frequency of 1. From further investigations, this page relates to *X. Henry Goodnough* an engineer and chief advocate for the creation of the *Quabbin Reservoir project*. However, the property relates to a person definition. This indicates an error present due to wrong mapping with Wikipedia Infobox keys. From the manual validation, the precision of the identified issues using the consistency measure accounts to 68%.

We have evaluated a total of 114 properties with consistency issues for *dbo:Place* class. We extracted all the data instances for the properties with consistency issues. From the manual inspection in *dbo:Place* class we identify data instances with erroneous conceptualization. For example, the property *dbo:weight* has 26 data instances mapped with *dbo:Place* type. We further investigate each of this data instances and corresponding Wikipedia pages. From manual investigation we can identity *dbo:weight* property erroneously mapped with *dbo:Place* type. Such as one of the data instance *wikipedia-en:Nokia_X5* is about mobile devices is mapped with *dbo:Place* type. This indicates an

inconsistency issue due to wrong schema presentation. Based on the manual validation results we evaluate precision of 76% for *dbo:Place* class.

7.3.3 Completeness

For the 3cixty KB, we analyzed the 2016-06-06 and 2016-09-09 releases; we evaluated the properties attached to *lode:Event* entities. DBpedia KB entity type of *foaf:Person* and *dbo:Place* in 201510 and 201604 releases has 131 and 437 properties with completeness issues. For manual validation, we manually inspected whether they are real issues.

3cixty. From the analysis of the 2016-06-06 and 2016-09-09 releases of the 3cixty KB releases, we found eight properties showing completeness issues. Based on the eight *lode:Event* class properties, we investigated all entities and attached properties. We first investigated five instances for each property, manually inspecting 40 different entities. From the investigation we observed that those entities that are presents in 2016-06-06 are missing in 2016-09-09 that leads to a completeness issue. Entities are missing in the 2016-09-09 release due to an error of the reconciliation algorithm. Based on this manual investigation, the completeness measure generates an output that has a precision of 95%.

DBpedia. We have randomly selected 50 properties from *foaf:Person* class which is identified as incomplete in the quantitative experiment. In our manual inspection, we investigated a small number of the subjects presented in each property. More specifically, we first checked five subjects for manual evaluation for each property. For DBpedia, we checked a total of 250 entities. For example, we identified that the property *bnfId* has completeness issue. We extracted all the subjects for the releases of 201510 and 201610.

In detail, the property *dbo:bnfId* for version 201604 has only 16 instances and for version 201510 has 217 instances. We performed a entities comparison between these two releases to identify the missing instances of the given property *dbo:bnfId* in the 201604 release. After a comparison between the two releases, we found 204 distinct instances missing in 201610 version of DBpedia. We perform a further manual investigation on the instances to verify the result.

One of the results of the analysis is *John_Hartley _(academic)*[4] who is available in the 201510 release. However, it is not found in 201604 release of DBpedia. To further validate such an output, we checked the source Wikipedia page using *foaf:primaryTopic* about *John Hartley (academic)*[5]. In the Wikipedia page *BNF ID* is present as linked to external source. In DBpedia from 201510 version to 201604 version update, this entity has been removed from the property *dbo:bnfId*. This example shows a completeness issue presents in the 201604 release of DBpedia for property *dbo:bnfId*.

Another example of DBpedia *foaf:Person*-type, properties with completeness issues are *dbo:firstRace* and *dbo:lastRace*. We extracted all the subjects present in the last two releases (201510 and 201604) and performed a set disjoint operation to identify the missing subjects. For manual validation, We first checked five subjects for the *dbo:firstRace* and *dbo:lastRace* property, checking a total of 250 entities. In the 201604 release, *dbo:firstRace* has 769 instances and in the 201510 release it has 777 instances. After the set disjoint operation between two releases (201510, 201604), we found 9 distinct instances missing in 201604 release of DBpedia EN. Furthermore, we manually inspected each instance to identify causes of incompleteness issue. One of the data instance *dbr:Bob_Said* for the *dbo:firstRace* property is available in the 201510 release. However, it is not present in 201604 release. We further explore the corresponding Wikipedia page using *foaf:primaryTopic*. In the Wikipedia page *firt race* is present as info box key. Due to DBpedia update from 201510 to 201604 version, this entity has been missing from the property *dbo:firstRace*. Similarly, we also found this entity is missing for the *dbo:lastRace* property. These examples present an ideal scenario for completeness issues in the 201604 release of the English version of DBpedia. Based on the manual inspection of 50 properties, we observed that completeness measure has the precision of 94%.

From the incomplete properties list of *dbo:Place* class we randomly selected 50 properties. We checked first five entities for manual evaluation. For *dbo:Place* class, we checked a total of 250 entities. For example, we identified that the property *dbo:parish* has completeness issue. We extracted all the instances for the releases of 201510 and 201610. Then we perform manual inspection for

[4] http://dbpedia.org/page/John_Hartley_(academic)
[5] https://en.wikipedia.org/wiki/John_Hartley_(academic)

each entity and compared with the Wikipedia sources to identify the causes of quality issues.

For example, property *dbo:parish* has 26 entities for 201510 and 20 entities in 201604. We collect missing resources after performing set disjoint operation. One of the results of the set disjoint operation is *Maughold_(parish)* missing in the 201604 version. To further validate such an output, we checked the source Wikipedia page using *foaf:primaryTopic* about *wikipedia-en:Maughold_(parish)*. In the Wikipedia page *Parish* is presented as title definition of the captain of parish militia. In particular, in DBpedia from 201510 version to 201604 version update, this entity has been removed from the property *dbo:parish*. Based on the investigation of the properties, we compute our completeness measure has the precision of 86%.

From the Spanish version of DBpedia, *dbo:Place* entity type completeness measure we found 3606 properties with completeness value of 0. This indicates a potential completeness issue present for these properties. From the 3606 property, we randomly select the property *dbo:prefijoTelefónicoNombre* for manual validation. We collected all the subjects (56109, 55387) from the two releases (201604, 201610). Then we performed a set of disjoint operations between two triples set to identify those triples missing from the 201610.

From the set disjoint operation, we found a total of 1982 subject missing from 201610 version. To keep the manual work in a feasible level, we selected a subset of 200 subjects for evaluation in a random manner. One of the results of the analysis is location *Morante*,[6] which is available in the 201604 release. However, it is missing in 201610 release of DBpedia. To further validate such an output, we checked the source Wikipedia page using *foaf:primaryTopic* about *Morante*[7]. In the Wikipedia page *prefijo TelefónicoNombre* is present in the infobox as key. In DBpedia ES from 201604 version to 201610 version update, this subject has been missing from the property *prefijo TelefónicoNombre*. This example shows a completeness issue presents in the 201610 release of DBpedia for property *prefijo TelefónicoNombre*. Based on the investigation over the subset of property values, we compute our completeness measure has the precision of 89%.

[6] http://es.dbpedia.org/page/Morante
[7] https://es.wikipedia.org/wiki/Morante

7.4 Validation using Integrity Constraints

The general goal of the constraints based validation is to evaluate the entity types using the constraints features from RDF shape induction based on predictive modeling. Examples of the constraints generation processes are presented in Section 5.1. From the quantitative analysis, we have identified multiple entity types and properties with quality issues. In particular, we selected the entity types from the quality analysis for RDF shape induction and evaluated the constraints classifier performance using machine learning model. Our approach has been implemented with a prototype written in R[8].

Because we are evaluating the constraints values as a classification problem, it is necessary to validate the annotations further and create a gold standard. In this context, we have manually inspected the constraints feature (Sec. 6.3.1) values from the 3cixty and DBpedia KB. However, to keep the manual inspection tasks at the feasible level, we have selected a subset of properties for an entity type.

7.4.1 Feature Extraction

In this section, we present examples of the SHACL based integrity constraints implementation and we report the results of *i)* Cardinality constraints, *ii)* Range Constraints, and *iii)* String constraints. We describe the above while reporting the analysis performed on the English DBpedia 201604 release.

Cardinality constraints: We generate cardinality information for each property associated with the instances of a given class. For example, by analyzing 1,767,272 *dbo:Person* instances in DBpedia, we extract the cardinality distribution for *dbo:Person-dbo:deathDate* as reported in Table 7.15.

During the feature extration step, this raw profiling data is used to derive a set of features that can be used for predicting the cardinality. Another example of cardinality distribution is reported in Table 7.16 for the *dbo:Sport/dbo:union* property.

At first we extract the raw cardinalities. Based on the raw values we compute the distinct cardinality values distributions similar to the one reported in

[8] https://github.com/rifat963/RDFShapeInduction

Table 7.15 Cardinality Counts for *dbo:Person-dbo:deathDate*.

Cardinality	**Instances**	**Precentage**
0	1,355,038	76.67%
1	404,069	22.87%
2	8,165	0.46%

Table 7.16 Cardinality Counts for *dbo:Sport/dbo:union*.

Cardinality	**Instances**	**Precentage**
0	1,662	84.88%
1	279	14.14%
2	10	0.05%
3	5	0.02%
4	2	0.01%

Table 7.16. Note that there are three distributions, one is the raw cardinalities (0,1,0,3,1,2,1,6,1,0), then distinct cardinalities (0,1,2,3,4) and finally one is the percentages of instances per each cardinality (84.88%, 14.24%, 0.05%, 0.02%, 0.01%). Further, for each of the three distributions we derive 30 statistical measures including min-max cardinalities, mean, mode, standard deviation, variance, quadratic mean, skewness, percentiles, and kurtosis[89].

Table 7.17 reports 30 features (P1 to P30) selected for a classifier that predicts the cardinality category with example values for the *dbo:Sport* class *dbo:union* property. Features P1 to P13 are related to raw cardinality distribution, features P14 to P20 are related to the distinct cardinality distribution, and features P21 to P30 are related to the percentages distribution. For example, P1 present a minimum cardinality value of 0 for *dbo:Sport/dbo:union* and P2 presents maximum that is 4. Our intuition is that these are descriptive to classify the cardinality category. Nevertheless, the data can be noisy and either min or/and max could be outliers. To address this we add statistical features that give more insights about the distribution of the cardinalities such as mean, mode, kurtosis, standard deviationsm, skewness, variance and four percentiles. Our motivation for using these statistical values is that each of these could provide some insights related to different possible cardinality distributions. Based on

the cardinality level, which is presented in Table 5.1, we create a gold standard by annotating the properties with corresponding constraints values and create the RDF Shape for validation. For instance, the *dbo:Person-dbo:deathDate* corresponding SHACL property constraints are generated as illustrated by Listing 5.2.

Range Constraints: We collected statistics about the number of IRIs, Literals, and Blank nodes for each property associated with instances of a given class as shown in Table 7.18. The blank node counts are also generated by the data collection stage but they are not reported because there were no blank nodes in this example.

Furthermore, we also explore object type information by analyzing all the IRI and blank node objects. Table 4 shows an example of object type information by analyzing all the objects of *dbo:Person/dbp:deathPlace* class-property combination. It contains the number of objects, the number of distinct of objects of each class type and their respective percentages. As it can be seen, the objects of *dbo:Person/dbp:deathPlace* are typed as many different classes. And, in general, it can be seen that most objects are typed with multiple classes (e.g., with equivalent classes, super classes). Also there are some objects that should not be associated (*i.e.*, inconsistent) with the *dbp:deathPlace* property, for example, a *Broadcaster* should not be a death place of a person. Further, there are some objects for which the type information is not available.

Similarly, for literal objects our data collection module extracts the information about their data types. Table 7.20 shows an example of extracted information for the class-property combination *dbp:Person/dbp:deathDate*. For each datatype, it shows the number of objects, number of distinct objects, and their corresponding percentages. Such an information provides heuristics about which should be the corresponding datatype.

String constraints: We use statistics about the literals to identify the minLength and maxLength of the String values. Based on the string length distribution of literal values, we explore the 1st quartile and 3rd quartile to identify the minimum and maximum length.

More specifically, we evaluate the interquartile range (IQR) based on the string length literal values of a property. For example, in Table 7.21, we report the string length distribution of the *foaf:Person* class *dbo:Title* property together

Table 7.17 *dbo:Sport/dbo:union* 30 statistical measures (p1 to p30) from raw cardinality estimation.

ID	Description	Example	ID	Description	Example
P1	Min Cardinality	0	P16	Distinct Quadratic Mean	2.4495
P2	Max Cardinality	4	P17	Distinct Kurtosis	-1.2
P3	Mean	0.16445	P18	Distinct Standard Deviation	1.5811
P4	Mode	0	P19	Distinct Skewness	0
P5	Quadratic mean	0.44972	P20	Distinct variance	2.5
P6	Kurtosis	13.7897	P21	Percentages Mins	0.0010
P7	Standard Deviation	0.41868	P22	Percentage Max	0.8488
P8	Skewness	3.09484	P23	0 Percentage	0.8488
P9	Variance	0.17529	P24	1 Percentage	0.1429
P10	98th percentile	1	P25	Percentage Mean	0.2
P11	2nd percentile	0	P26	Percentage Quad. Mean	0.3849
P12	75nd percentile	0	P27	Percentage Kurtosis	0.3849
P13	25th percentile	0	P28	Percentage Standard Deviation	0.3677
P14	Distinct Cardinalities	5	P29	Percentage Skewness	2.0948
P15	Distinct Mean Card.	0	P30	Percentage Variance	0.1352

with frequency of string length. Similarly, in Table 7.22, it is illustrated the *dbo:BirthName* property frequency distribution.

In this example, both properties have a small central tendency towards the mean. Our main focus is to identify a range of minLength and maxLength

Table 7.18 Object node type information.

Class-property	IRI		Literals	
	Total	Distinct	Total	Distinct
dbo:Person/dbp:birthPlace	89,355	21,845	44,639	20,405
dbo:Person/dbp:name	21,496	15,746	115,848	100,931
dbo:Person/dbp:deathDate	127	111	65,272	32,449
dbo:Person/dbp:religion	8,374	786	6,977	407

Table 7.19 Classes of *dbo:Person-dbp:birthPlace* objects.

Object Class	Objects (89,355)		Distinct Objects (21,845)	
	Count	%	Count	%
schema:Place	71,748	80.29	16,502	75.54
dbo:Place	71,748	80.29	16,502	75.54
dbo:PopulatedPlace	71,542	80.07	16,353	74.86
dbo:Settlement	41,216	46.13	14,184	64.93
other rows are omitted for brevity				
schema:Product	2	00.00	2	00.01
dbo:Broadcaster	2	00.00	2	00.01
Unknown	9,790	10.95	2,888	13.22

Table 7.20 Datatypes of *dbp:Person-dbp:deathDate* literals.

Datatype	Objects (65,272)		Distinct Objects (32,449)	
	Count	%	Count	%
xsd:date	39,761	60.92	26,726	82.36
xsd:integer	13,543	20.75	1,758	5.42
rdf:langString	6,388	9.79	3,512	10.82
xsd:gMonthDay	5,446	8.34	366	1.13
dt:second	113	0.17	66	0.20
xsd:double	20	0.03	20	0.06
dt:hour	1	0.00	1	0.00
Total	65,272	100	32,449	100

Table 7.21 Frequency distribution of *foaf:Person/dbo:Title* property.

String Length	Frequency	Percentage
16	20	31.25 %
13	7	10.93%
15	5	7.81%
other rows are omitted for brevity		
20	4	6.25%

for literal objects. In this account, we use the interquartile range for *dbo:title* to identify minLength and maxLength. We used the 3rd quartile (Q3) of the string length as maxLength and the 1st quartile (Q1) as minLength for the *dbo:title* property. In Figure 7.8, we present a boxplot of the *dbo:title* property. In particular, using the interquartile range, we can present the string range constraints as a binary classifier.

Table 7.22 Frequency distribution of *foaf:Person/dbo:BirthName* property.

String Length	Frequency	Percentage
20	32	13.14 %
21	26	10.65%
19	25	10.24%
other rows are omitted for brevity		
22	17	6.96%

7.4.2 Model Preparation

In this experimental analysis for English DBpedia KB, we used the expected cardinalities for 174 properties (associated with an instance of a given class). Also, we collected a subset of 200 properties associated with the *dbo:Place* entity type for IRI objects and the datatype for literal objects. On the other hand, for Spanish DBpedia we collected expected cardinality for 240 properties and 219 properties for the range constraints based on the *dbo:Organization* entity type. Furthermore, we collected

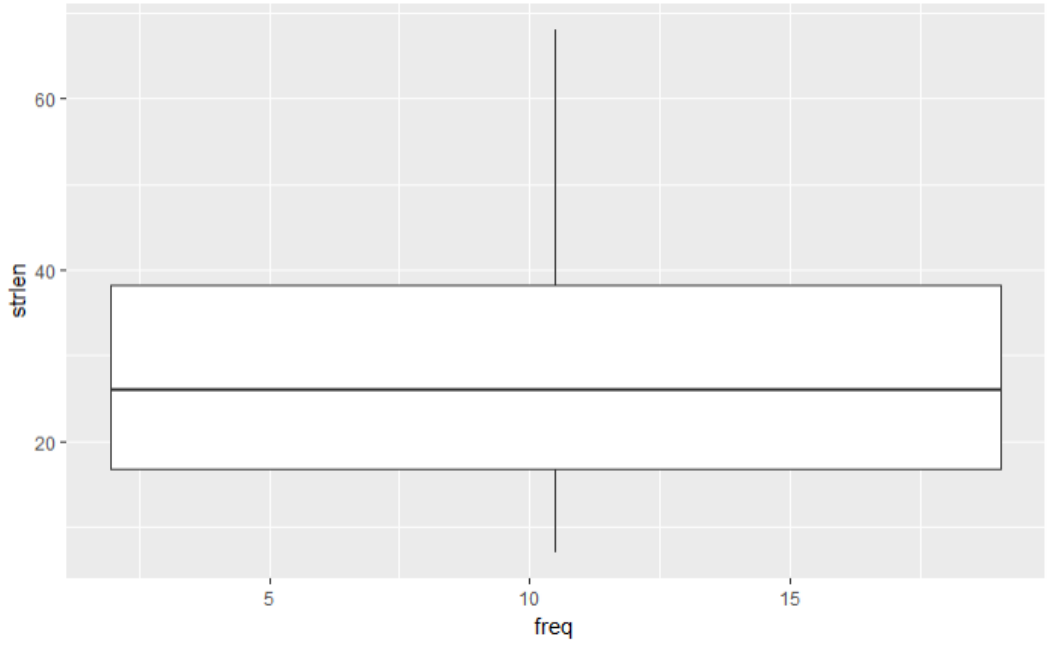


Fig. 7.8 *foaf:Person* class *dbo:title* property string length box plot.

dataset with cardinality information for each property associated with instances of a given class for 215 properties for the 3cixty Nice KB. Similarly, for range constraints, we collected 215 properties associated with IRI and the datatype for literal objects. In particular, we evaluated each dataset by manually inspecting properties annotated with integrity constraints to create the gold standards. We considered this partial gold standard as the training dataset.

From the initial analysis of the datasets, we found that the minimum cardinality value has an imbalance in the classifier distribution. We observed that rare events occur in case of selected constraints as response variables where variation between two variables is less than 15%. We applied SMOTE (Synthetic Minority Over-sampling Technique) [92] for oversampling the rare events. The SMOTE function over-samples response variables by using bootstrapping and k-Nearest Neighbor to synthetically create additional observations of that response variable. In our experiment, we applied an over-sampling value of 100 to double the number of positive cases, and an undersampling value of 200 to keep half of what was created as negative cases. It balances the classifier and achieves better performance than only under-sampling the majority class. The results are reported in Table 7.23. After applying the SMOTE technique, we applied 10-fold cross-validation based on the

learning models mentioned in Section 2.8 and the classifiers use default parameters unless otherwise stated.

We evaluate each dataset using the simplest classifier (known as ZeroR [93]) to establish the baseline value that must be enhanced by our model. More specifically, we used the ZeroR [93] classifier for range, minimum and maximum cardinality. It is a simple classification method which relies on the target and ignores all predictors. ZeroR based on predicting the most-frequent class in target variable. For example, DBpedia minimum cardinality training set has two class MIN0 and MIN1+. As the number of the MIN0 feature is 215 and the MIN1+ feature is 65. The ZeroR classifier will be based on MIN0. In the Table 7.24 we present baseline accuracy for minimum and maximum cardinality for both datasets using ZeroR. Although there is no predictability power in ZeroR, it is useful for determining a baseline performance as a benchmark for other classification methods.

Table 7.23 DBpedia and 3cixty Nice distribution of cardinality constraints.

Distribution	**Minimum Cardinality**		**Maximum Cardinality**		**Range Constraint**	
	MIN0	MIN1+	MAX1	MAX1+	IRI	LIT
3cixty Nice KB						
Without SMOTE	47%	52.8%	79.2%	20.8%	68.7%	31.3%
With SMOTE (100,200)	50%	50%	50%	50%	50%	50%
English DBpedia KB						
Without SMOTE	76.5%	23.5%	53%	47%	71.5%	28.5%
With SMOTE(100,200)	50%	50%	50%	50%	50%	50%
Spanish DBpedia KB						
Without SMOTE	72%	28%	56%	44%	69.4%	30.6%
With SMOTE(100,200)	50%	50%	50%	50%	50%	50%

Within this context, at first, we explored the classifier accuracy for each dataset. Classifier accuracy gives an insight into the performance using the ratio of the number of correct predictions out of all predictions made and presented as a percentage where 100% is the best an algorithm can achieve. In Table 7.25 we present the percentage of cardinality classifier accuracy for DBpedia KB and 3cixty KB. From Table 7.25, most of the machine learning algorithm has a high percentage of accuracy compared to baseline algorithm. For example, in the case of DBpedia KB, both minimum

Table 7.24 Baseline accuracy (using ZeroR) for 3cixty KB and DBpedia KB.

Knowledge Base	Minimum Cardinality	Maximum Cardinality	Range Constraint
3cixty	53.5%	80%	78.2%
English DBpedia	71.2%	81.6%	79.8%
Spanish DBpedia	76.4%	52.9%	72.3%

and maximum cardinality classifier have a high percentage of accuracy such as Random Forest model has 97.6% for minimum cardinality and 97.03% for maximum cardinality. However, 3cixty KB minimum and maximum cardinality have significant variation in classifier performance. Such as, Naive Bayesian (58.3%) has near to baseline accuracy of 53.5% for minimum cardinality. However, Random forest model has a higher percentage of accuracy of 74.8%. In this context, random forest model gives higher accuracy for almost all five models. We further evaluated our constraints classifier performance using the measures introduced in Section 2.8.

Table 7.25 Classifier accuracy for the DBpedia KB and 3cixty KB

Knowledge Base		Random Forest	Least Squares SVM	Multilayer Perceptron	K-Nearest Neighbour	Naive Bayes
3cixty	Minimum Cardinality	74.89%	66.67%	61.11%	63.89%	58.3%
	Maximum Cardinality	81.52%	76.29%	79.89%	79.31%	69.44%
	Range Constraint	82.57%	78.23%	83.17%	75.48%	71.68%
English DBpedia	Minimum Cardinality	97.61%	94.81%	95.11%	92.64%	87.83
	Maximum Cardinality	97.03%	82.67%	79.81%	83.41%	84.04%
	Range Constraint	91.81%	73.52%	72.22%	71.84%	70.58%
Spanish DBpedia	Minimum Cardinality	82.97%	79.36%	84.56%	74.87%	72.67%
	Maximum Cardinality	85.34%	73.74%	78.72%	76.81%	72.36%
	Range Constraint	83.74%	72.22%	71.43%	69.47%	76.37%

7.4.3 Model Evaluation

In detail, the model evaluation results using precision, recall and F1 measures for the constraints classifiers are mentioned below.

3cixty Nice. In Table 7.26, we present the 3cixty KB three constraints classifier performance measures. Overall ensemble algorithm, Random Forest model achieved a greater than 90% F1 value for all three classifiers. More specifically, for minimum cardinality, random forest model reached 91% F1 score where it achieved 96% precision. On the other hand, Neural Network of multilayer perceptron algorithm reached 90% F1 score for range constraints. However, simple Naive Bayes learning algorithm has significantly lower F1 ($<$70%) score compared to all the classifiers F1 scores. K-Nearest Neighbour (K-NN) has the lowest F1 score for the maximum cardinality and range constraints.

English DBpedia. Table 7.26 illustrates the three classifiers performance measures for the English version of DBpedia KB. Similar to 3cixty KB, ensemble learning algorithm, Random Forest proven to be effective in achieving greater than 90% F1 value for all three classifiers. Moreover, minimum cardinality constraints random forest algorithm reached 97% F1 score where it achieved 98% precision. However, in case of minimum cardinality classifier, other learning algorithms such as Neural Network and Least Squares SVM also achieved greater than 90% F1 score.

Spanish DBpedia. Table 7.28 illustrates the integrity constraints performance measure for the Spanish DBpedia Dataset. Compared to other models, Random Forest achieved the highest F1 score for all three classifiers. Moreover, it achieved 92.85% F1 score for maximum cardinality classifier. On the other hand, compared to random forest model, Least Squares SVM achieved also achieved the F1 score of 87.23% for the minimum cardinality classifier. However, for Spanish DBpedia, Naive Bayes classifier has the lowest F1 score for all the constraints.

Table 7.26 Integrity Constraints performance measures for 3cixty KB.

Learning Algorithm	Minimum Cardinality			Maximum Cardinality			Range		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Random Forest	**0.9626**	**0.8729**	**0.9156**	**0.8909**	**0.9423**	**0.9159**	**0.9333**	**0.9032**	**0.9180**
Multilayer Perceptron	0.8812	0.8812	0.8128	0.8113	0.8269	0.8190	0.9375	0.8823	0.9091
Least Squares SVM	0.7692	0.7263	0.7471	0.8070	0.8846	0.8440	0.8148	0.9167	0.8627
Naive Bayes	0.7152	0.6932	0.7040	0.7288	0.8268	0.7748	0.8266	0.7462	0.8275
K-Nearest Neighbour	0.6991	0.6695	0.6840	0.7049	0.8269	0.7611	0.7837	0.8285	0.8055

Table 7.27 Integrity Constraints performance measure for English DBpedia.

Learning Algorithm	Minimum Cardinality			Maximum Cardinality			Range		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Random Forest	**0.9890**	**0.9574**	**0.9730**	**0.9842**	**0.9920**	**0.9881**	**0.9457**	**0.9527**	**0.9594**
Least Squares SVM	0.9944	0.9468	0.9700	0.8491	0.9574	0.9000	0.8596	0.9231	0.8902
Multilayer Perceptron	0.9674	0.9468	0.9570	0.8167	0.9601	0.8826	0.8262	0.8657	0.8456
K-Nearest Neighbour	0.9511	0.9309	0.9409	0.8797	0.8750	0.8773	0.8361	0.8425	0.8393
Naive Bayes	0.9401	0.8351	0.8845	0.9065	0.7739	0.8350	0.8953	0.7951	0.8422

Table 7.28 Integrity Constraints performance measure for Spanish DBpedia.

Learning Algorithm	Minimum Cardinality			Maximum Cardinality			Range		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Random Forest	**0.8971**	**0.8547**	**0.8754**	**0.9247**	**0.9323**	**0.9285**	**0.8741**	**0.8954**	**0.8846**
Least Squares SVM	0.8517	0.8940	0.8723	0.8070	0.8846	0.8440	0.8348	0.8416	0.8381
Multilayer Perceptron	0.8670	0.8183	0.8419	0.8863	0.8517	0.8685	0.7942	0.7701	0.7819
K-Nearest Neighbour	0.8378	0.8170	0.8272	0.8168	0.7901	0.8032	0.7714	0.7808	0.7761
Naive Bayes	0.7091	0.7278	0.7183	0.7862	0.7961	0.7911	0.7620	0.7901	0.7758

Chapter 8

Discussion and Limitations

This chapter reports a synthesis of the main findings of the experimental analysis presented in Chapter 7. Taking into consideration of quality evaluation, we potentially detected errors in various stages of evolving KBs. However, we only explored small subsets of the datasets for manual evaluation. Due to the unrestrained evolution of KBs, quality measures could lead to increasing number of false positive results. In this context, we introduced integrity constraints based validation approach. Figure 8.1 illustrates the main results of this thesis, labeled A to G.

More specifically, Section 8.1 discusses the evaluation results of the proposed quality assessment approach. Section 8.2 addresses the performance of the learning models. Based on the experimental analysis, Section 8.3 outlines the impact of KB changes using two use cases and Section 8.4 presents an initial observation on quality issues of literal values. Section 8.5 introduces an approach to lifespan analysis for evolving KBs. Finally, Section 8.6 outlines the limitations of the proposed approach.

8.1 Evolution Analysis to Drive Quality Assessment

Similarly to Radulovic et al. [94], we present a discussion of our approach with respect to the following four criteria:

Conformance provides insights to what extent a quality framework and characteristics meet established standards. In our approach, we have proposed four quality characteristics. Among them we selected the completeness and consis-

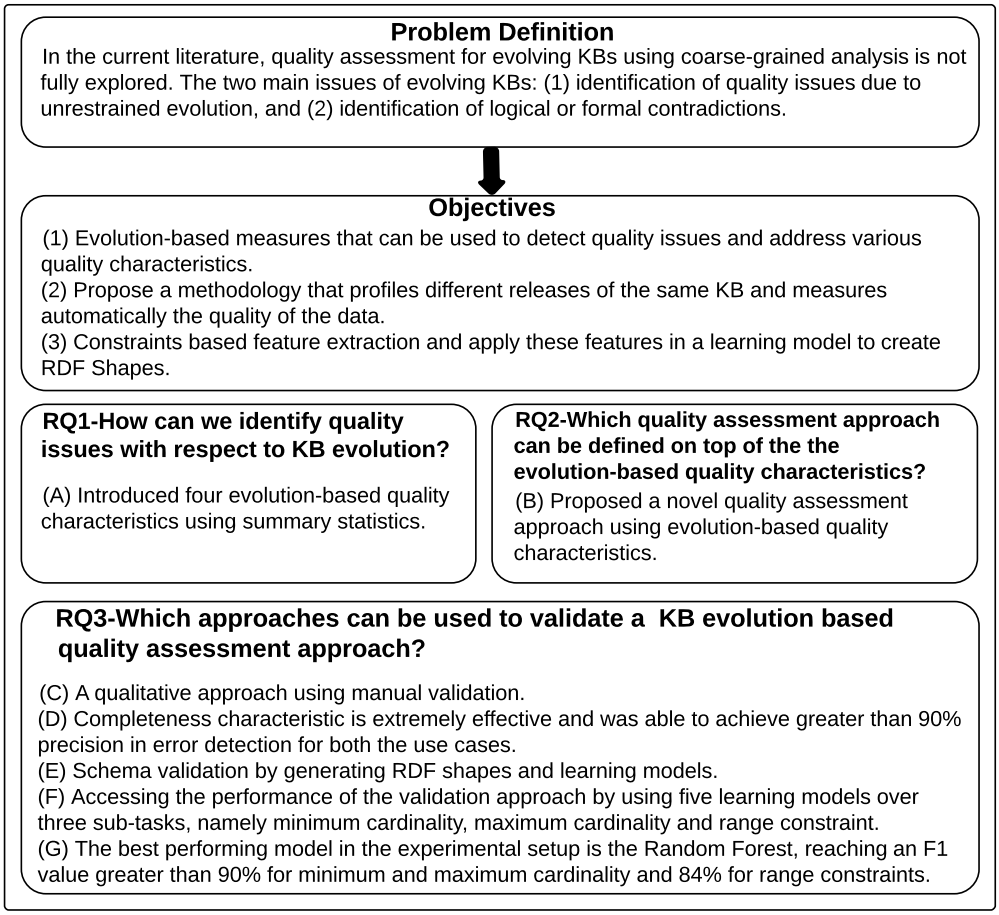


Fig. 8.1 Summary of the main results of the Quality Assessment and Validation Approach.

tency quality characteristics according to the guidelines from the ISO 25012 standard. On the other hand, we followed the study presented by Ellefi *et al.* [9] to propose persistency and historical persistency quality characteristics.

Applicability implies the practical aspects of the quality assessment approach. In general, coarse-grained analysis significantly improves the space and time complexity regarding data analysis. We envision that our approach can be automated using daily snapshot generations and automatically creating periodic reports. We experimented with two different knowledge bases and verified our hypothesis for both. Our implementation follows a simple structure and it can scalable to KBs with a large number of entities and properties.

Causes of Quality Issues provides insights regarding detected issues using our approach. In our approach, we identified two types of quality issues: *i)* errors in the data source extraction process, and *ii)* erroneous schema presentation. In the case of the 3cixty Nice KB, we only found issues based on the data source extraction process. For example, we found a significant number of resources missing in the last release of *lode:Event* class due to algorithmic error. On the other hand, the 3cixty Nice KB schema remains unchanged in all KB releases. More specifically, we did not find any real issues based on the schema presentation. In the case of the DBpedia KB, we found both types of quality issues. For example, entities missing in *foaf:Person* class due to incorrect mapping of field values in the data extraction process. For example, we found a significant number of resources missing due to wrong schema presentation for the DBpedia KB. Such a property *dbo:Lake* is mapped with *foaf:Person*-type due to automatic mapping with wrong Wikipedia infobox keys. Based on the two use cases, our approach has proven highly efficient to identify quality issues in the data extraction and integration process.

Performance measures how much the output generated by our approach is accurated by counting true positives (TP) or false positives (FP). Using persistency and historical persistency measures, we analyzed the KB changes with each release to detect any quality issues. For each use cases, we have only detected if the persistency measure leads to TP or FP results. Based on the qualitative analysis, the persistency measure for the 3cixty Nice KB *lode:Event* class has quality issues in the last release(2016-09-09). In the case of the DBpedia KB *dbo:Species* class, we did not find any quality issues on the last release(201604) which lead to FP results. On the other hand, we have evaluated precision based on completeness and consistency measures for the both KBs. Overall, we evaluated the property completeness measure in terms of precision through manual evaluation. Considering computational complexity, we only use count and difference operation for measurement functions. We assume that our computational complexity will be $O(N_T)$ where the N_T is the total number of entities for type T. The computed precision of completeness measure in our approach is: *i)* 94% for *foaf:Person*-type entities of the English DBpedia KB; *ii)* 89% for *dbo:Place*-type entities of the Spanish DBpedia KB, and *iii)* 95% for the *lode:Event*-type entities of the 3cixty Nice KB. We only identify consistency issue in case of DBpedia and computed precision of 68% through

manual evaluation for *foaf:Person*-type entities and 76% for *dbo:Place*-type entities.

8.2 Modeling Performance

The goal of the validation using integrity constraints is twofold: *(i)* evaluating the performance of the cardinality and range constraints classifier using learning models, and *(ii)* creating constraints dataset that can be used for RDF shape generation.

In this context, the prediction performance of constraint classifiers are measured by precision, recall and F1 score. Overall, our constraints classifiers achieved high predictive performance with the Random Forest model. For example, the Random Forest cardinality classifiers achieved the highest F1 score for all the KBs. Furthermore, the Multilayer Perceptron and the Least Squares SVM also achieved high F1 scores greater than 90% for the English DBpedia. Concerning the range constraints, we explored the object node type constraint for each property associated with a given class. Similar to cardinality constraints, Random Forest algorithm achieved a high F1 score of 95.94% for the English DBpedia KB. This makes the consistency evaluation approach adaptable and facilitate adoption for multiple KBs.

Furthermore, we applied a Naive Bayes classifier. The model provides apriori probabilities of no-recurrence and recurrence events as well as conditional probability tables across all attributes. We considered Naive Bayes as a baseline model to explore the classifier performance compared to other learning algorithms. In this context, other models achieved better performance values compared to the Naive Bayes learning algorithm.

Finally, we generate constraints once the constraint prediction models are built. Based on the Random Forest model, we created the constraints datasets. More specifically, we combined all the constraints related to a given class and, for each, we generate an RDF Shape. An example of the RDF Shape in SHACL for the *foaf:Person* class is illustrated in 8.1 using cardinality and range constraints. Furthermore, we perceived that the generated constraints datasets can be used in other tools such as RDFUnit [50]. We considered this extension of our RDF shape induction approach as a future work.

Listing 8.1 DBpedia Person SHACL Shape

```
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .

ex:DBpediaPerson a sh:NodeShape;
 sh:targetClass foaf:Person;
# node type Literal
 sh:property [sh:path foaf:name;
  sh:minCount 1;
  sh:nodeKind sh:Literal ];
# for MIN1 and MAX1 cardinality
 sh:property [ sh:path dbo:birthDate;
  sh:datatype xsd:date ;
  sh:minCount 1;
  sh:maxCount 1;
  sh:nodeKind sh:Literal ] ;
# node type IRI
 sh:property [sh:path dbp:birthPlace;
  sh:nodeKind sh:IRI;
  sh:or ( [sh:class schema:Place]
    [ sh:class dbo:Place ] )
  ];
# node type literal
 sh:property [ sh:path dbp:deathDate;
  sh:nodeKind sh:Literal;
  sh:datatype xsd:date ] .
```

8.3 Frequency of Knowledge Base Changes

KBs can be classified according to application areas, schema changes, and frequency of data updates. The two KB we analyzed, namely 3cixty Nice and DBpedia, fall into two distinct categories: *i)* continuously changing KB with high frequency updates (daily updates), and *ii)* KB with low frequency updates (monthly or yearly updates).

i) KBs continuously grow because of an increase in the number of instances and predicates, while they preserve a fixed schema level (T-Box). These KBs are usually

available via a public endpoint. For example DBpedia Live [1] and 3cixty Nice KB falls in this category. In fact, the overall ontology remains the same but new triples are added as effect of new information being generated and added to the KB. In our analysis, we collected batches of data at nearly fixed time intervals for 8 months.

ii) KBs grow at intervals since the changes can be observed only when a new release is deployed. DBpedia is a prime example of KBs with a history of releases. DBpedia consists of incremental versions of the same KB where instances and properties can be both added or removed and the schema is subjected to changes. In our approach we only considered subject changes in a KB over all the releases. In particular, we only considered those triples T from common classes $(c_1...c_i)$ or properties $(p_1...p_i)$ presented in all releases $(V_1....Vn)$ of the same KB.

8.4 Quality Assessment of Literal Values

We performed experimental analysis based on each quality characteristics. From the quantitative analysis, we identified properties with quality issues from consistency and completeness measures. We validated the observed results through manually investigating each properties value. From our investigation, we perceive that those properties that have quality issues may contain an error in literal values. We then further investigated our assumption in the case of DBpedia. We choose one random property of the *foaf:Person*-type entities. We finally examined the literal values to identify any error present.

From our quantitative analysis on the completeness characteristics of DBpedia, we detected the property *dbo:bnfId* triggered a completeness issue. Only 16 resources in DBpedia 201604 version had such an issue, while 217 resources in 201510 version. We, therefore, further investigated the property *dbo:bnfId* in details on the 201604 release. We explored the property description that leads to Wikidata link[2] and examined how *BnF ID* is defined. It is an identifier for the subject issued by BNF (Bibliothèque nationale de France). It is formed by 8 digits followed by a check digit or letter. In Table 8.1, we present 6 subjects and objects of 207 *bnfId* property where each object follows the formatting structure. However, the literal value for

[1] http://wiki.dbpedia.org/online-access/DBpediaLive
[2] https://www.wikidata.org/wiki/Property:P268

subject *Quincy_Davis_(musician)*[3] contains a "/" between the digits "12148" and "cb16520477z", which does not follow standard formatting structure issued by BNF (Bibliothèque nationale de France). It clearly points to an error for the subject *Quincy_Davis_(musician)*.

From the initial inspection, we assume that it can be possible to identify an error in any literal value using our approach. However, to detect errors in literal values, we need to extend our quality assessment framework to inspect literal values computationally. We considered this extension of literal value analysis as a future research endeavour.

Table 8.1 A sample of 6 subjects and objects of *bnfId* property

Subject	**Object**
dbp:Tom_Morello	"14051227k"
dbp:David_Kherdian	"14812877"
dbp:Andrè_Trocmè	"cb12500614n"
dbp:Quincy _Davis_(musician)	"12148/cb16520477z"
dbp:Charles_S. _Belden	"cb140782417"
dbp:Julien_Durand _(politician)	"cb158043617"

8.5 Lifespan Analysis of Evolving KBs

On the basis of the dynamic feature [9], a further conjecture poses that the growth of the knowledge in a mature KB ought to be stable. From our analysis on the 3cixty Nice and the DBpedia KB, we observed that variations in the knowledge base growth could affect quality issues. Furthermore, we argue that quality issues can be identified through monitoring lifespan of an RDF KBs.

We can measure growth level of KB resources (instances) by measuring changes presented in different releases. In particular, knowledge base growth can be measured by detecting the changes over KB releases utilizing trend analysis such as the use of simple linear regression. Based on the comparison between observed and predicted

[3] http://dbpedia.org/resource/Quincy_Davis_(musician)

values, we can detect the trend in the KB resources, thus detecting anomalies over KB releases if the resources have a downward trend over the releases. Following, we derive KB lifespan analysis regarding change patterns over time as well as experiments on the 3cixty Nice KB and the DBpedia KB. To measure the KB growth, we applied linear regression analysis of entity counts over KB releases. In the regression analysis, we checked the latest release to measure the normalized distance between an actual and a predicted value. In particular, in the linear regression we used entity count (y_i) as dependent variable and time period (t_i) as independent variable. Here, $n = total\ number\ of\ KB\ releases$ and $i = 1...n$ present as the time period.

We start with a linear regression fitting the count measure of the class (C):

$$y = at + b$$

The residual can be defined as:

$$residual_i(C) = a \cdot t_i + b - count_i(C)$$

We define the normalized distance as:

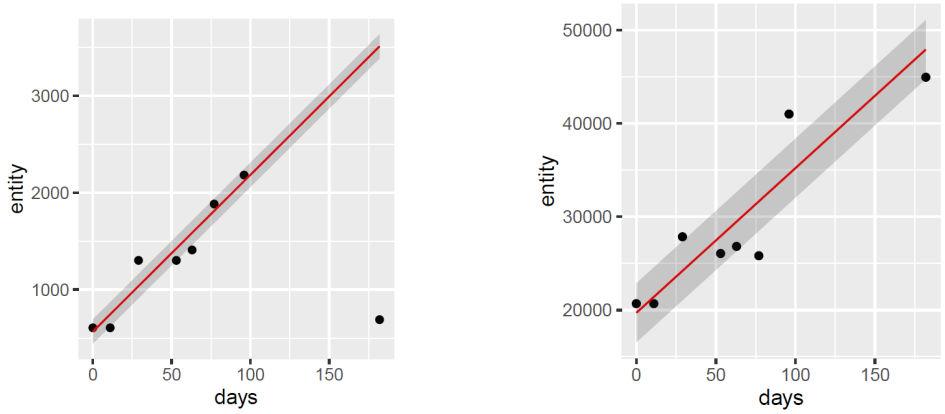
$$ND(C) = \frac{residual_n(C)}{mean(|residual_i(C)|)}$$

Based on the normalized distance, we can measure the KB growth of a class C as:

$$KBgrowth(C) = \begin{cases} 1 & if ND(C) \geq 1 \\ 0 & if ND(C) < 1 \end{cases}$$

More specifically, the value is 1 if the normalized distance between actual value is higher than the predicted value of type C otherwise it is 0. In particular, if the KB growth measure has the value of 1 then the KB may have an unexpected growth with unwanted entities otherwise the KB remains stable.

3cixty Nice case study The experimental data is reported in Table 7.2a. We applied the linear regression over the eight releases for the *lode:Event*-type and *dul:Place*-type entities. We present the regression line in Figure 8.2a and 8.2b.



(a) lode:Event (b) dul:Place

Fig. 8.2 3cixty two classes KB growth measure

From the linear regression, the 3cixty Nice has a total of $n = 8$ releases where the 8^{th} predicted value for *lode:Event* $y'_{event_8} = 3511.548$ while the actual value=689. Similarly, for *dul:Place* $y'_{place_8} = 47941.57$ and the actual value=44968.

The residuals, e_{events_8}= $|689 - 3511.548| = 2822.545$ and e_{places_8}= $|44968 - 49741.57| = 2973.566$. The mean of the residuals, $e_{event_i} = 125.1784$ and $e_{place_i} = 3159.551$, where $i = 1...n$.

So the normalized distance for, 8^{th} *lode:Event* entity $ND_{event} = \frac{2822.545}{125.1784} = 22.54818$ and *dul:Place* entity $ND_{place} = \frac{2973.566}{3159.551} = 0.9411357$.

For the *lode:Event* class, $ND_{events} \geq 1$ so the KB growth measure value = 1. However, for the *dul:Place* class, $ND_{places} < 1$ so the KB growth measure value =0 .

In the case of 3cixty Nice KB, the *lode:Event* class clearly presents anomalies as the number of distinct entities drops significantly on the last release. In Figure 8.2a, the *lode:Event* class growth remains constant until it has errors in the last release. It has higher distance between actual and predicted value based on the *lode:Event*-type entity count. However, in the case of *dul:Place*-type, the actual entity count in the last release is near to the predicted value. We can assume that on the last release the 3cixty Nice KB has improved the quality of data generation matching the expected growth.

DBpedia Case study The experimental data is reported in Table 7.1. Based on the KB growth measure definition, we measured the normalized distance for each class (Table 8.2). We compared with the number of entities from the last release

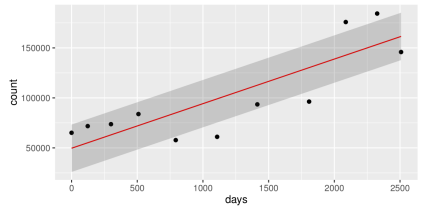
(201604) actual value and predicted value from the linear regression to measure the normalized distance. From the results observed for *dbo:Artist*, *dbo:Film*, and *dbo:MusicalWork*, the normalized distance is near the regression line with $ND < 1$. In Figure 8.3, we present the DBpedia 10 classes KB growth measure value and we can observe that there is no issue in the KB.

For instance while inspecting the different trends over the KB releases and calculating the normalized distance, we identified that *foaf:Person*-type last release (201604) entity count has a higher growth (over the expected). Such as *foaf:Person* has KB growth measure of 1 where normalized distance, $ND = 2.08$. From this measure we can implies that, in *foaf:Person* there is persistency issue. We can imply that additions in a KB can also be an issue. It can include unwanted subjects or predicates.

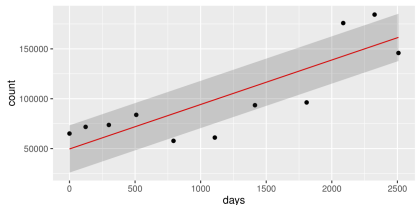
Table 8.2 DBpedia 10 class Summary

Class	**Normalized Distance(ND)**	**KB Growth measure**
dbo:Animal	3.05	1
dbo:Artist	0.66	0
dbo:Athlete	2.03	1
dbo:Film	0.91	0
dbo:MucsicalWork	0.56	0
dbo:Organisation	2.02	1
dbo:Place	5.03	1
dbo:Species	5.87	1
dbo:Work	1.05	1
foaf:Person	2.08	1

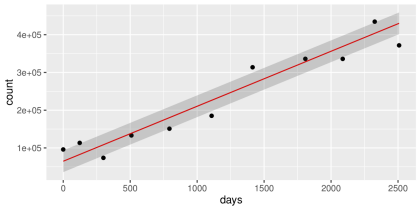
We define this KB growth measure as *stability characteristic*. A simple interpretation of the stability of a KB is monitoring the dynamics of knowledge base changes. This measure could be useful to understand high-level changes by analyzing KB growth patterns. Data curators can identify persistency issues in KB resources using lifespan analysis. However, a further exploration of the KB lifespan analysis is needed, and we consider this as a future research activity.



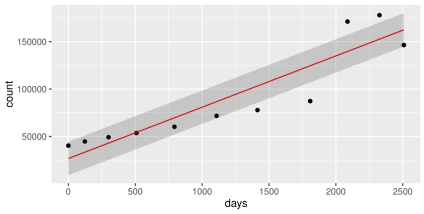
(a) dbo:Animal



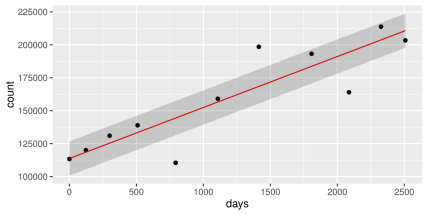
(b) dbo:Artist



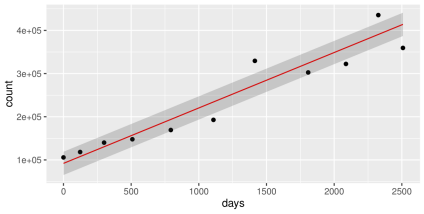
(c) dbo:Athlete



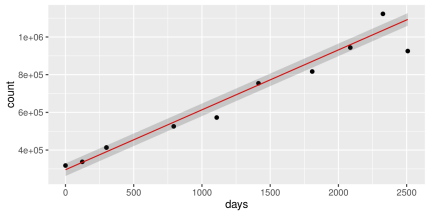
(d) dbo:Film



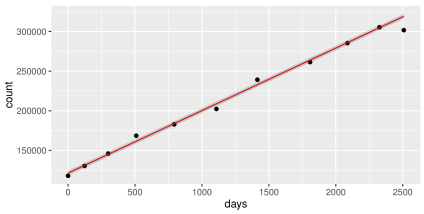
(e) dbo:MusicalWork



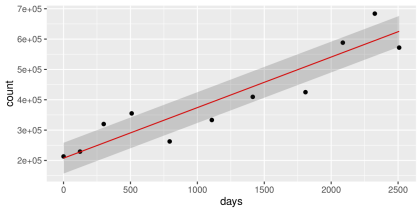
(f) dbo:Organization



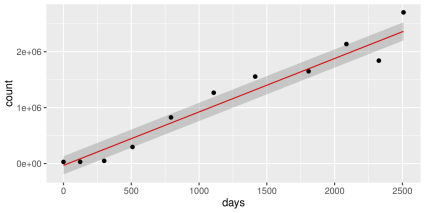
(g) dbo:Place



(h) dbo:Species



(i) dbo:Work



(j) foaf:Person

Fig. 8.3 DBpedia 10 classes KB growth measure

8.6 Limitations

We have identified the following three main limitations.

First, as a basic measurement element, we only considered aggregated measures from statistical profiling such as frequency of properties in a class. For the qualitative analysis, we considered raw knowledge base differences among releases. In order to detect actual differences, we would need to store two releases of a KB in a single graph, and perform the set difference operation. We performed manual validation by inspecting data sources. However, this approach of qualitative analysis has several drawbacks from a technical point of view. Furthermore, regardless of the technical details, the set difference operation is, computationally-wise, extremely expensive. As a future work, we plan to extend our manual validation approach by cross-referencing GitHub issues or mailing lists.

Second, KBs are growing over time with new resources that are added or deleted. In this study, we only considered the negative impact of erroneous deletion of resources. As a future work, we plan to investigate the negative impact of the erroneous addition of resources in the KBs.

Third, in the experimental analysis of validation using integrity constraints, we have adopted the manual evaluation strategy in order to create the training dataset. In this context, the training set depends on the specific KB, and the evaluation of the annotations require considerable domain knowledge to decide if a constraint is correct or incorrect. As a future work, we plan to extend our partial gold standard creation strategy by using data mining approach or automatic validation using schema definitions.

Chapter 9

Conclusions and Future Work

The main motivation for the work presented in this thesis is rooted in the concepts of Linked data dynamics [9] on the one side and knowledge base quality on the other side. We focused on automated shape validation as well as on automating the timely process of quality issue detection without user intervention using KB evolution analysis. Knowledge about Linked Data dynamics is essential for a broad range of applications such as effective caching, link maintenance, and versioning [10]. However, less focus has been given towards understanding knowledge base resource changes over time to detect anomalies over various releases. To verify our assumption, we proposed four quality characteristics, based on the evolution analysis.

We proposed a quality assessment approach by profiling quality issues using different Knowledge Base (KB) releases. In particular, we explored the benefits of aggregated measures using quality profiling. The advantage of our approach lies in the fact that it captures anomalies for an evolving KB that can trigger alerts to the data curators in the quality repairing processes. More specifically, we consider coarse-grained analysis as an essential requirement to capture any quality issues for an evolving KB. Although coarse-grained analysis cannot detect all possible quality issues, it helps to identify common quality issues such as erroneous deletion of resources in the data extraction and integration processes. Moreover, it addresses the main drawback of fine-grained analysis using raw change detection, the significant space and time complexity. Since, we perceive that if the KB has design issues, our quality assessment approach might lead to increase the number of false positives. We

introduced an RDF validation approach using integrity constraints based on SHACL representation to further verify the results.

We conclude this thesis by providing an overview of the main contributions along with the answers to the research questions introduced in Chapter 1 (Section 1.2). Furthermore, we present the future directions in which we intend to solve the limitations and move forward with the research conducted in the specific areas.

9.1 Summary of Contributions

In this section, we revisit each research questions and summarize the solutions for the problems identified at the beginning of this thesis (Section 1.1).

RQ1: Evolution-based quality characteristics. The research question we aimed to answer is:

- *RQ1: How can we identify quality issues with respect to KB evolution?*

To answer this question, at first, an empirical study is performed to identify the quality issues present in the evolving KBs (Chapter 2). In this study, three main quality issues of an evolving knowledge base is explored (Chapter 2.6): *(i)* Lack of consistency, *(ii)* Lack of completeness, and *(iii)* Lack of persistency. Furthermore, the issues of quality assessment in the evolving KBs is explored using a comprehensive survey of current approaches about linked data dynamics, knowledge base quality assessment and validation (Chapter 3). As shown in chapter 3, there are significant approaches present for KB quality assessment and validation, however, less focus is given towards benefit of KB dynamicity.

In this context, the factors affecting the KB evolution and data profiling dynamic features is explored in Chapter 4. Using the dynamic features of the data profiling, four evaluation-based quality characteristics are proposed (Chapter 4.3): Persistency, Historical Persistency, Consistency, and Completeness. The lack of persistency issues are explored by using persistency and historical persistency characteristics. The proposed persistency and historical persistency quality characteristics use the degree of change feature of the dataset

dynamics. Furthermore, completeness and consistency quality characteristics is proposed from the ISO 25012 standard, and the measurement function is based on the history of a KB update. The quality measures are based on simple statistical analysis using entity count and property frequency for an entity type. These quality characteristics can be applied to any knowledge base, and it is demonstrated for two different use cases, namely 3cixty Nice and DBpedia (Chapter 7).

RQ2: Evolution-based quality assessment approach. The research question we aimed to answer is:

- *RQ2: Which quality assessment approach can be defined on top of the the evolution-based quality characteristics?*

In response to this question, a quality assessment approach is proposed using evolution analysis and validation using RDF shape induction based on predictive modeling (Chapter 6). In this work, *quality assessment* is considered as the process of statistically assessing a KB using evolution analysis. On the contrary, *validation* as a mean to evaluate the performance of the quality evaluation process. Within this context, RDF validation approach is introduced using SHACL based constraints shape induction which relies on the data profiling information (Chapter 5). In particular, using the evolution-based quality characteristics and RDF shape induction approach, a KB quality assessment approach (KBQ) is proposed that explore the KB changes over various releases of the same KB (Chapter 6).

The approach provides an assessment of the overall quality characteristic and is not aimed at pinpointing the individual issues in the KB, but it aims to identify potential problems in the data processing pipeline. Such an approach produces a smaller number of coarse-grained issue notifications that are directly manageable without any filtering and provide useful feedback to data curators. This approach can be applied to any knowledge base, and it is demonstrated for two different use cases, namely 3cixty Nice and DBpedia (Chapter 7). The proposed approach can provide a quality problem report to KB curators. Furthermore, as a proof of concepts, we have created KBQ, a tool for KB quality assessment and validation using evolution-based quality characteristics (Chapter 6).

RQ3: Validation approaches leveraging on quality characteristics and integrity constraints. The research question we aimed to answer is:

- *Which approaches can be used to validate a KB evolution based quality assessment approach?*

In order to address this research question, the proposed approach is applied to two different knowledge bases of different size and semantics, and its operations verified using empirical analysis (Chapter 7). Besides, since this approach uses the first stage of statistical profiling, it reduces the search space of the suspicious issues, which are then verified by the learning models (Sec.7.4). Thus, it can be applied to also larger knowledge bases (Chapter 8).

From the experimental analysis applied on two different KBs, three causes of quality issues are identified (Chapter 8): *(i)* errors in the data source extraction process, *(ii)* erroneous conceptualization, and *(iii)* error in object type. In the case of the 3cixty Nice KB, only issue is found due to error in the data source extraction process. On the contrary, all three types of quality issues are found for DBpedia KB. Also, significant number of issues are identified due to wrong schema presentation for the DBpedia KB. From the empirical investigation, we perceive that those properties that have quality issues may contain an error in literal values. Based on the two use cases, the proposed approach has proven to be highly effective to identify quality issues in the data extraction and integration process.

The experimental analysis is based on four quality characteristics: persistency, historical persistency, consistency, and completeness. The analysis of Persistency and Historical Persistency shows that KB with periodic update (3cixty Nice KB) could lead to detect missing values. Missing values could happen due to algorithm error as in the case of 3cixty Nice KB *lode:Event* class. On the contrary, KB with long duration of updates (DBpedia KB) issues do not always indicate actual errors. For example, *dbo:Species* subjects with the wrong type in version 201510 fixed in version 201604. From the quantitative and qualitative analysis of consistency measure, quality issues only found in the case of DBpedia KB. For example, no logical or formal contradiction found in the case of 3cixty Nice KB. In this context, periodicity of KB update could affect consistency issue. Continuously changing KBs with high-frequency updates (daily updates) such as the 3cixty Nice KB, has less quality issues

considering consistency quality characteristic. On the contrary, KBs with low-frequency updates (monthly or yearly updates), such as DBpedia KB, have more logical or formal contradictions based on consistency analysis. In the experimental analysis, completeness quality characteristics demonstrates extremely good performances for both 3cixty Nice and DBpedia KB. The completeness measure was able to detect actual issues with very high precision – 95% for the 3cixty Nice KB *lode:Event* class and 94% for the DBpedia *foaf:Person* class.

Proposed consistency analysis is extended using a profiling-based RDF shape induction approach and predictive modeling. RDF shape induction approach is tested based on cardinality constraints and range constraints. In this analysis, the performance of the five learning models are empirically assessed and the best performing model is identified according to the F1 score. In this context, among the five learning model, the best performing model is the Random Forest for both KBs. The approach reaches an F1 score greater than 90% with DBpedia datasets for cardinality constraints using Random Forest model. Nevertheless, the proposed approach is defined in a generic and flexible manner which can be extended to other types of constraints. In general, all learning models have good performances meaning that the problem is well configured and the features are predictive (Chapter 8).

9.2 Future Work

In this section, we describe the future work with regards to the main contributions of this thesis.

Quality Characteristics. We plan to expand our evolution based quality analysis approach by analyzing other quality characteristics presented in literature such as Zaveri *et al.* [22]. Also, we intend to apply our approach to KBs in other domain to further verify our assumption. Moreover, for consistency characteristics, we chose 100 as threshold value since from the empirical analysis at property level it allowed to maximize the precision of the approach. However, the process of threshold value selection needs further investigation. As a future work, we plan to perform additional statistical analysis to motivate the choice of this threshold.

Impact of addition of resources. A limitation of the current approach is that we only considered the negative impact of deletion of resources. We plan to study how we can dynamically adapt impact of the addition of resources in a KB. Within this context, we explored the lifespan analysis of evolving KBs. We argue that quality issues can be identified through monitoring lifespan of a KB. This has led to conceptualize the Stability quality characteristics, which is meant to detect anomalies in a KB. We plan to monitor various KB growth rates to validate this assumption. In particular, we want to explore further *(i)* which factors are affecting stability characteristics and *(ii)* validating the stability measure.

RDF Shape induction in KBQ tool. We plan to integrate the RDF Shape induction process presented in this thesis to KBQ-tool, in order to perform all validation approaches in a single tool which can cover different validation tasks. Furthermore, we plan to extend our validation approaches using automatic snapshots generation and to publish quality problem report in a triple format.

Literal Analysis. From the initial experiments, we assume that it can be possible to identify an error in literal value using our approach. We want to extend our quality assessment approach to inspect literal values;

Schema based Validation. We presented experimental analysis using three SHACL constraints types: cardinality, range, and string. As future work, we plan to extend our implementations to remaining SHACL constraints. As an output of our approach, we generate SHACL integrity constraints. We envision that these constraints can be applied to other tools such as RDFUnit [50] as a direct input. However, in RDFUnit they considered constraints in the form of RDFS/OWL axioms. We considered extending our approach to RDFUnit as future research work to favor the interoperability.

Furthermore, in our experimental analysis, we involved a human annotator to validate the datasets in order to create the partial gold standards. As future work, we plan to extend our evaluation strategy with an alternative approach such as the validation using OWL schema. However, it is challenging to explore an OWL schema for validation tasks. For example, DBpedia KB 201610 version ontology lacks axioms about cardinality constraints (owl:cardinality, owl:minCardinality, maxCardinality). The only information that we can extract from the ontology is indirectly using the axioms that define functional

properties (i.e., MAX 1 constraints). In this context, we plan to extend our approach to other KBs which contain complete OWL schema representations.

References

[1] ISO/IEC. 25012:2008 – software engineering – software product quality requirements and evaluation (square) – data quality model. Technical report, ISO/IEC, 2008.

[2] Rashid Mohammad, Torchiano Marco, Giuseppe Rizzo, Mihindukulasooriya Nandana, and Corcho Óscar. A Quality Assessment Approach for Evolving Knowledge Bases. *Semantic Web*, 2018.

[3] Thomas Gottron and Christian Gottron. Perplexity of Index Models over Evolving Linked Data. In Valentina Presutti, Claudia d'Amato, Fabien Gandon, Mathieu d'Aquin, Steffen Staab, and Anna Tordai, editors, *The Semantic Web: Trends and Challenges*, pages 161–175, Cham, 2014. Springer International Publishing.

[4] Jeremy Debattista, Christoph Lange, Sören Auer, and Dominic Cortis. Evaluating the Quality of the LOD Cloud: An Empirical Investigation. *Semantic Web*, 2017.

[5] Giri Kumar Tayi and Donald P. Ballou. Examining Data Quality. *Communications of the ACM*, 41(2):54–57, February 1998.

[6] Renata Dividino, Ansgar Scherp, Gerd Gröner, and Thomas Grotton. Change-a-LOD: Does the Schema on the Linked Data Cloud Change or Not? In Hartig Olaf, Sequeda Juan, Hogan Aidan, and Matsutsuka Takahide, editors, *Proceedings of the Fourth International Workshop on Consuming Linked Data (COLD2013) co-located with the 12th International Semantic Web Conference (ISWC 2013)*, Volume 1034 of CEUR Workshop Proceedings, pages 87–98, Sydney, Australia, 2013. CEUR-WS. org.

[7] Chifumi Nishioka and Ansgar Scherp. Information-theoretic Analysis of Entity Dynamics on the Linked Open Data Cloud. In *Proceedings of the 3rd International Workshop on Dataset Profiling and Federated Search for Linked Data (PROFILES '16) co-located with the 13th ESWC 2016 Conference*, Volume 1597 of CEUR Workshop Proceedings, Anissaras, Greece, 2016. CEUR-WS. org.

[8] Vicky Papavasileiou, Giorgos Flouris, Irini Fundulaki, Dimitris Kotzinos, and Vassilis Christophides. High-level Change Detection in RDF(S) KBs. *ACM Transactions on Database Systems (TODS)*, 38(1):1:1–1:42, April 2013.

[9] Mohamed Ben Ellefi, Zohra Bellahsene, J Breslin, Elena Demidova, Stefan Dietze, Julian Szymanski, and Konstantin Todorov. RDF Dataset Profiling - a Survey of Features, Methods, Vocabularies and Applications. *Semantic Web*, pages 1–29, 2018.

[10] Tobias Käfer, Ahmed Abdelrahman, Jürgen Umbrich, Patrick O'Byrne, and Aidan Hogan. Observing Linked Data Dynamics. In Philipp Cimiano, Oscar Corcho, Valentina Presutti, Laura Hollink, and Sebastian Rudolph, editors, *The Semantic Web: Semantics and Big Data*, pages 213–227, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[11] Jack E. Olson. *Data Quality: The Accuracy Dimension*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2003.

[12] Felix Naumann. Data profiling revisited. *SIGMOD Rec.*, 42(4):40–49, February 2014.

[13] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[14] Nandana Mihindukulasooriya, María Poveda-Villalón, Raúl García-Castro, and Asunción Gómez-Pérez. Collaborative Ontology Evolution and Data Quality - An Empirical Analysis. In *OWL: Experiences and Directions – Reasoner Evaluation: 13th International Workshop, OWLED 2016, and 5th International Workshop, ORE 2016, Bologna, Italy, November 20, 2016, Revised Selected Papers*, pages 95–114, 2017.

[15] Nathalie Pernelle, Fatiha Saïs, Daniel Mercier, and Sujeeban Thuraisamy. RDF data evolution: efficient detection and semantic representation of changes. In *Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS'16)*, Volume 1695 of CEUR Workshop Proceedings. CEUR-WS. org, 2016.

[16] Max Völkel and Tudor Groza. SemVersion: An RDF-based Ontology Versioning System. In Miguel Baptista Nunes, editor, *Proceedings of IADIS International Conference on WWW/Internet (IADIS 2006)*, pages 195–202, Murcia, Spain, October 2006.

[17] Marios Meimaris, George Papastefanatos, Christos Pateritsas, Theodora Galani, and Yannis Stavrakas. A Framework for Managing Evolving Information Resources on the Data Web. *Computing Research Repository(CoRR)*, abs/1504.06451, 2015.

[18] Aidan Hogan, Andreas Harth, Alexandre Passant, Stefan Decker, and Axel Polleres. Weaving the Pedantic Web. In Bizer Christian, Heath Tom, Berners-Lee Tim, and Hausenblas Michael, editors, *3rd International Workshop on*

Linked Data on the Web (LDOW2010), in conjunction with 19th International World Wide Web Conference, Volume 628 of CEUR Workshop Proceedings, Raleigh, USA, 2010. CEUR-WS. org.

[19] Jose Emilio Labra Gayo, Eric Prud'hommeaux, Iovka Boneva, and Dimitris Kontokostas. *Validating RDF Data*, volume 7 of *Synthesis Lectures on the Semantic Web: Theory and Technology*. Morgan & Claypool Publishers LLC, sep 2017.

[20] Nandana Mihindukulasooriya, María Poveda-Villalón, Raúl García-Castro, and Asunción Gómez-Pérez. Collaborative Ontology Evolution and Data Quality - An Empirical Analysis. In Mauro Dragoni, María Poveda-Villalón, and Ernesto Jimenez-Ruiz, editors, *OWL: Experiences and Directions – Reasoner Evaluation*, pages 95–114, Cham, 2017. Springer International Publishing.

[21] Antoine Isaac and Riccardo Albertoni. Data on the Web Best Practices: Data Quality Vocabulary. W3C note, W3C, December 2016. https://www.w3.org/TR/2016/NOTE-vocab-dqv-20161215/.

[22] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer. Quality Assessment for linked Data: A Survey. *Semantic Web*, 7(1):63–93, 2016.

[23] Giuseppe Rizzo Nandana Mihindukulasooriya Mohammad Rashid, Marco Torchiano and Oscar Corcho. A Quality Assessment Approach for Evolving Knowledge Bases. *Journal of Web Semantics*, 2017.

[24] Rashid Mohammad, Rizzo Giuseppe, Mihindukulasooriya Nandana, Torchiano Marco, and Corcho Óscar. KBQ - A Tool for Knowledge Base Quality Assessment Using Evolution Analysis. In Tiddi Ilaria, Rizzo Giuseppe, and Corcho Óscar, editors, *Proceedings of Workshops and Tutorials of the 9th International Conference on Knowledge Capture (K-CAP2017)*, Volume 2065 of CEUR Workshop Proceedings, Austin, Texas, 2017. CEUR-WS. org.

[25] Giuseppe Rizzo, Raphaël Troncy, Oscar Corcho, Anthony Jameson, Julien Plu, Juan Carlos Ballesteros Hermida, Ahmad Assaf, Catalin Barbu, Adrian Spirescu, Kai-Dominik Kuhn, Irene Celino, Rachit Agarwal, Cong Kinh Nguyen, Animesh Pathak, Christian Scanu, Massimo Valla, Timber Haaker, Emiliano Sergio Verga, Matteo Rossi, and José Luis Redondo Garcia. 3cixty@Expo Milano 2015 enabling visitors to explore a smart city. In *ISWC 2015, 14th International Semantic Web Conference, Semantic Web Challenge, October 11-15, 2015, Bethlehem, PA, USA*, Bethlehem, UNITED STATES, 10 2015.

[26] Nandana Mihindukulasooriya, Mohammad Rifat Ahmmad Rashid, Giuseppe Rizzo, Raúl García-Castro, Oscar Corcho, and Marco Torchiano. RDF Shape Induction Using Knowledge Base Profiling. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, pages 1952–1959, New York, NY, USA, 2018. ACM.

[27] Denny Vrandečić and Markus Krötzsch. Wikidata: A Free Collaborative Knowledge base. *Commun. ACM*, 57(10):78–85, September 2014.

[28] Krzysztof Goczyła, Aleksander Waloszek, and Wojciech Waloszek. Contextualization of a DL knowledge base. In *20th International Workshop on Description Logics (DL2007),Brixen-Bressanone, near Bozen-Bolzano, Italy 8–10 June, 2007*, Volume 250 of CEUR Workshop Proceedings. CEUR-WS. org, 2007.

[29] Heiko Paulheim. Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods. *Semantic Web*, 8(3):489–508, 2017.

[30] Raphaël Troncy, Giuseppe Rizzo, Anthony Jameson, Oscar Corcho, Julien Plu, Enrico Palumbo, Juan Carlos Ballesteros Hermida, Adrian Spirescu, Kai-Dominik Kuhn, Catalin Barbu, Matteo Rossi, Irene Celino, Rachit Agarwal, Christian Scanu, Massimo Valla, and Timber Haaker. 3cixty: Building comprehensive knowledge bases for city exploration. *Web Semantics: Science, Services and Agents on the World Wide Web*, 46-47:2 – 13, 2017.

[31] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the ACM International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, New York, NY, USA, 2008. ACM.

[32] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. DBpedia–A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[33] Nuno Laranjeiro, Seyma Nur Soydemir, and Jorge Bernardino. A Survey on Data Quality: Classifying Poor Data. In *IEEE 21st Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 179–188. IEEE, Nov 2015.

[34] Sören Auer, Jens Lehmann, and Axel-Cyrille Ngonga Ngomo. Introduction to Linked Data and Its Lifecycle on the Web. In *Proceedings of the 7th International Conference on Reasoning Web: Semantic Technologies for the Web of Data*, RW'11, pages 1–75, Berlin, Heidelberg, 2011. Springer-Verlag.

[35] Jeremy Debattista, Sören Auer, and Christoph Lange. Luzzu - A Methodology and Framework for Linked Data Quality Assessment. *Journal of Data and Information Quality (JDIQ)*, 8(1):4:1–4:32, October 2016.

[36] T. Groza, A. Oellrich, and N. Collier. Using silver and semi-gold standard corpora to compare open named entity recognisers. In *IEEE International Conference on Bioinformatics and Biomedicine*, pages 481–485. IEEE, Dec 2013.

[37] Ning Kang, Erik M. van Mulligen, and Jan A. Kors. Training text chunkers on a silver standard corpus: can silver replace gold? *BMC Bioinformatics*, 13(1):17, Jan 2012.

[38] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2016.

[39] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989.

[40] Pedro Domingos and Michael Pazzani. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29(2):103–130, Nov 1997.

[41] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, Jan 1991.

[42] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[43] Bernhard Pfahringer. *Random model trees: an effective and scalable regression method*. University of Waikato, Department of Computer Science, 2010.

[44] Marina Sokolova and Guy Lapalme. A Systematic Analysis of Performance Measures for Classification Tasks. *Information Processing and Management: an International Journal*, 45(4):427–437, July 2009.

[45] Jürgen Umbrich, Stefan Decker, Michael Hausenblas, Axel Polleres, and Aidan Hogan. Towards dataset dynamics: Change frequency of linked open data sources. In *Proceedings of the WWW2010 Workshop on Linked Data on the Web(LDOW)*, Volume 628 of CEUR Workshop Proceedings, Raleigh, USA, 2010. CEUR-WS. org.

[46] Jürgen Umbrich, Boris Villazón-Terrazas, and Michael Hausenblas. Dataset dynamics compendium: A comparative study. In *Proceedings of the First International Workshop on Consuming Linked Data (COLD2010) at the 9th International Semantic Web Conference (ISWC2010)*, Volume 665 of CEUR Workshop Proceedings, Shanghai, China, 2010. CEUR-WS. org.

[47] Michel Klein, Dieter Fensel, Atanas Kiryakov, and Damyan Ognyanov. Ontology versioning and change detection on the web. In Asunción Gómez-Pérez and V. Richard Benjamins, editors, *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, pages 197–212, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[48] Christian Bizer and Richard Cyganiak. Quality-driven Information Filtering Using the WIQA Policy Framework. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(1):1–10, January 2009.

[49] Pablo N. Mendes, Hannes Mühleisen, and Christian Bizer. Sieve: Linked Data Quality Assessment and Fusion. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, EDBT-ICDT '12, pages 116–123, New York, NY, USA, 2012. ACM.

[50] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. Test-driven Evaluation of Linked Data Quality. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 747–758, New York, NY, USA, 2014. ACM.

[51] Jeremy Debattista, Santiago Londoño, Christoph Lange, and Sören Auer. Quality Assessment of Linked Datasets Using Probabilistic Approximation. In Fabien Gandon, Marta Sabou, Harald Sack, Claudia d'Amato, Philippe Cudré-Mauroux, and Antoine Zimmermann, editors, *The Semantic Web. Latest Advances and New Domains*, pages 221–236, Cham, 2015. Springer International Publishing.

[52] Jeremy Debattista, Christoph Lange, and Sören Auer. A Preliminary Investigation Towards Improving Linked Data Quality Using Distance-Based Outlier Detection. In Yuan-Fang Li, Wei Hu, Jin Song Dong, Grigoris Antoniou, Zhe Wang, Jun Sun, and Yang Liu, editors, *Semantic Technology*, pages 116–124, Cham, 2016. Springer International Publishing.

[53] André Melo and Heiko Paulheim. Detection of Relation Assertion Errors in Knowledge Graphs. In *Proceedings of the Knowledge Capture Conference*, K-CAP 2017, pages 22:1–22:8, New York, NY, USA, 2017. ACM.

[54] Maribel Acosta, Amrapali Zaveri, Elena Paslaru Bontas Simperl, Dimitris Kontokostas, Fabian Flöck, and Jens Lehmann. Detecting Linked Data quality issues via crowdsourcing: A DBpedia study. *Semantic Web*, 9:303–335, 2018.

[55] Dimitris Kontokostas, Amrapali Zaveri, Sören Auer, and Jens Lehmann. TripleCheckMate: A Tool for Crowdsourcing the Quality Assessment of Linked Data. In Pavel Klinov and Dmitry Mouromtsev, editors, *Knowledge Engineering and the Semantic Web*, pages 265–272, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[56] Volker Nannen. Quality Characteristics of Linked Data publishing data sources. Master's thesis, Humboldt-Universität of Berlin, Berlin, Germany, 2010.

[57] Ahmad Assaf, Raphaël Troncy, and Aline Senart. Roomba: An Extensible Framework to Validate and Build Dataset Profiles. In Fabien Gandon, Christophe Guéret, Serena Villata, John Breslin, Catherine Faron-Zucker, and Antoine Zimmermann, editors, *The Semantic Web: ESWC 2015 Satellite Events*, pages 325–339, Cham, 2015. Springer International Publishing.

[58] A. Rula, M. Palmonari, and A. Maurino. Capturing the Age of Linked Open Data: Towards a Dataset-Independent Framework. In *2012 IEEE Sixth International Conference on Semantic Computing*, pages 218–225, Sept 2012.

[59] Christian Fürber and Martin Hepp. SWIQA - A Semantic Web information quality assessment framework. In *Proceedings of the 19th European Conference on Information Systems (ECIS 2011)*, volume 15, page 19, 2011.

[60] Magnus Knuth, Dimitris Kontokostas, and Harald Sack. Linked Data Quality: Identifying and Tackling the Key Challenges. In *Proceedings of the 1st Workshop on Linked Data Quality co-located with 10th International Conference on Semantic Systems (SEMANTiCS)*, Volume 1215 of CEUR Workshop Proceedings, Leipzig, Germany, 2014. CEUR-WS. org.

[61] Suzanne M Embury, Binling Jin, Sandra Sampaio, and Iliada Eleftheriou. On the Feasibility of Crawling Linked Data Sets for Reusable Defect Corrections. In Magnus Knuth, Dimitris Kontokostas, and Harald Sack, editors, *Proceedings of the 1st Workshop on Linked Data Quality co-located with 10th International Conference on Semantic Systems (SEMANTiCS)*, Volume 1215 of CEUR Workshop Proceedings, Leipzig, Germany, 2014. CEUR-WS. org.

[62] Heiko Paulheim and Christian Bizer. Improving the Quality of Linked Data Using Statistical Distributions. *Int. J. Semant. Web Inf. Syst.*, 10(2):63–86, April 2014.

[63] Huiying Li, Yuanyuan Li, Feifei Xu, and Xinyu Zhong. Probabilistic error detecting in numerical linked data. In Qiming Chen, Abdelkader Hameurlain, Farouk Toumani, Roland Wagner, and Hendrik Decker, editors, *Database and Expert Systems Applications*, pages 61–75, Cham, 2015. Springer International Publishing.

[64] Edna Ruckhaus, Maria-Esther Vidal, Simón Castillo, Oscar Burguillos, and Oriana Baldizan. Analyzing Linked Data Quality with LiQuate. In Valentina Presutti, Eva Blomqvist, Raphael Troncy, Harald Sack, Ioannis Papadakis, and Anna Tordai, editors, *The Semantic Web: ESWC 2014 Satellite Events*, pages 488–493, Cham, 2014. Springer International Publishing.

[65] Grigoris Antoniou and Frank van Harmelen. *Web Ontology Language: OWL*, pages 67–92. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[66] Boris Motik, Ian Horrocks, and Ulrike Sattler. Bridging the gap between OWL and relational databases. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2):74 – 89, 2009.

[67] Jiao Tao, Evren Sirin, Jie Bao, and Deborah L McGuinness. Extending OWL with Integrity Constraints. In Haarslevand Volker, Toman David, and Weddell Grant, editors, *International Workshop on Description Logics (DL)*, Volume 573 of CEUR Workshop Proceedings, Waterloo, Ontario, Canada, 2010. CEUR-WS. org.

[68] Peter F. Patel-Schneider. Using Description Logics for RDF Constraint Checking and Closed-world Recognition. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 247–253. AAAI Press, 2015.

[69] Heiko Paulheim and Heiner Stuckenschmidt. Fast Approximate A-Box Consistency Checking Using Machine Learning. In Harald Sack, Eva Blomqvist, Mathieu d'Aquin, Chiara Ghidini, Simone Paolo Ponzetto, and Christoph Lange, editors, *The Semantic Web. Latest Advances and New Domains*, pages 135–150, Cham, 2016. Springer International Publishing.

[70] Ziawasch Abedjan and Felix Naumann. Improving RDF Data Through Association Rule Mining. *Datenbank-Spektrum*, 13(2):111–120, Jul 2013.

[71] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[72] Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2007.

[73] Hassan Khosravi and Bahareh Bina. A survey on statistical relational learning. In *Canadian Conference on AI*, pages 256–268. Springer, 2010.

[74] Lorenz Bühmann, Daniel Fleischhacker, Jens Lehmann, Andre Melo, and Johanna Völker. Inductive Lexical Learning of Class Expressions. In Krzysztof Janowicz, Stefan Schlobach, Patrick Lambrix, and Eero Hyvönen, editors, *Knowledge Engineering and Knowledge Management*, pages 42–53, Cham, 2014. Springer International Publishing.

[75] Adrien Basse, Fabien Gandon, Isabelle Mirbel, and Moussa Lo. DFS-based frequent graph pattern extraction to characterize the content of RDF Triple Stores. In *Web Science Conference 2010 (WebSci10)*, Raleigh, United States, April 2010.

[76] Ziqi Zhang, Anna Lisa Gentile, Eva Blomqvist, Isabelle Augenstein, and Fabio Ciravegna. Statistical Knowledge Patterns: Identifying Synonymous Relations in Large Linked Datasets. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web – ISWC 2013*, pages 703–719, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[77] Philippe Flajolet and G. Nigel Martin. Probabilistic Counting Algorithms for Data Base Applications. *Journal of Computer and System Sciences*, 31(2):182–209, September 1985.

[78] P. Flajolet. On Adaptive Sampling. *Computing*, 43(4):391–400, February 1990.

[79] Kyu-Young Whang, Brad T. Vander-Zanden, and Howard M. Taylor. A Linear-time Probabilistic Counting Algorithm for Database Applications. *ACM Trans. Database Syst.*, 15(2):208–229, June 1990.

[80] Stefan Heule, Marc Nunkesser, and Alexander Hall. HyperLogLog in Practice: Algorithmic Engineering of a State of the Art Cardinality Estimation Algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 683–692, New York, NY, USA, 2013. ACM.

[81] Thomas Neumann and Guido Moerkotte. Characteristic Sets: Accurate Cardinality Estimation for RDF Queries with Multiple Joins. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 984–994, Washington, DC, USA, 2011. IEEE Computer Society.

[82] Yannis Roussakis, Ioannis Chrysakis, Kostas Stefanidis, Giorgos Flouris, and Yannis Stavrakas. A Flexible Framework for Understanding the Dynamics of Evolving RDF Datasets. In Marcelo Arenas, Oscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan, Krishnaprasad Thirunarayan, and Steffen Staab, editors, *The Semantic Web - ISWC 2015*, pages 495–512, Cham, 2015. Springer International Publishing.

[83] Emanuel Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

[84] Richard A. Davis, Keh-Shin Lii, and Dimitris N. Politis. *Remarks on Some Nonparametric Estimates of a Density Function*, pages 95–100. Springer New York, New York, NY, 2011.

[85] Joakim Lindblad. Histogram Thresholding using Kernel Density Estimates. In *In Proceedings of the Swedish Society for Automated Image Analysis (SSAB) Symposium on Image Analysis, Halmstad, Sweden*, pages 41–44, 2000.

[86] Serge Abiteboul, Richard Hull, and Victor Vianu, editors. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1995.

[87] Stephen W. Liddle, David W. Embley, and Scott N. Woodfield. Cardinality constraints in semantic data models. *Data & Knowledge Engineering*, 11(3):235 – 270, 1993.

[88] Holger Knublauch and Dimitris Kontokostas. W3C Shapes Constraint Language (SHACL), July 2017.

[89] David A Freedman. *Statistical models: theory and practice*. cambridge university press, 2009.

[90] N. Mihindukulasooriya, G. Rizzo, R. Troncy, O Corcho, and R. Garcia-Castro. A Two-Fold Quality Assurance Approach for Dynamic Knowledge Bases: The 3cixty Use Case. In H. Paulheim, J. Lehmann, S. vatek, C. Knoblock, Horridge M., Lambrix P, and Parsia B, editors, *International Workshop on Completing and Debugging the Semantic Web (ESWC'16)*, Heraklion, Greece, 05 2016.

[91] Nandana Mihindukulasooriya, María Poveda-Villalón, Raúl García-Castro, and Asunción Gómez-Pérez. Loupe-An Online Tool for Inspecting Datasets in the Linked Data Cloud. In *Proceedings of the ISWC 2015 Posters & Demonstrations Track co-located with the 14th International Semantic Web Conference (ISWC-2015)*, Volume 1486 of CEUR Workshop Proceedings, Bethlehem, USA, 2015. CEUR-WS. org.

[92] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16(1):321–357, June 2002.

[93] WEKA. Weka manual for version 3-7-8. Technical report, WEKA, 2013.

[94] Filip Radulovic, Raúl García-Castro, and Asunción Gómez-Pérez. SemQuaRE - An Extension of the SQuaRE Quality Model for the Evaluation of Semantic Technologies. *Comput. Stand. Interfaces*, 38(C):101–112, February 2015.

[95] Rashid Mohammad, Rizzo Giuseppe, Mihindukulasooriya Nandana, Torchiano Marco, and Corcho Óscar. Knowledge Base Evolution Analysis: A Case Study in the Tourism Domain. In *Proceedings of Workshops on Knowledge Graphs on Travel and Tourism co-located with 18th International Conference on Web Engineering (ICWE)*, Caceres,Spain, 2018.

[96] Mohammad Rashid and Marco Torchiano. A systematic literature review of open data quality in practice. In *Proceedings of 2nd Open Data Research Symposium (ODRS)*, Madrid, Spain, 2016.

[97] Rashid Mohammad, Torchiano Marc, Rizzo Giuseppe, Mihindukulasooriya Nandana, and Corcho Oscar. Completeness and Consistency Analysis for Evolving Knowledge Bases(Under Review). *Journal of Web Semantics*, 2018.

[98] M. Rashid, L. Ardito, and M. Torchiano. Energy Consumption Analysis of Algorithms Implementations. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–4. IEEE, Oct 2015.

[99] Mohammad Rashid, Luca Ardito, and Marco Torchiano. Energy Consumption Analysis of Image Encoding and Decoding Algorithms. In *Proceedings of the Fourth International Workshop on Green and Sustainable Software*, GREENS '15, pages 15–21, Piscataway, NJ, USA, 2015. IEEE Press.

Appendix A

User Interfaces for the KBQ Tool and Data Extraction REST APIs

This appendix illustrates examples of the web based interfaces and the data extraction REST APIs for KBQ-tool. The application is available at [1]. A recorded video of KBQ in action for two KBs namely, DBpedia and 3cixty is available at [2].

KBQ is a tool that helps you to perform quality analysis on any Knowledge Base (KB) using four quality characteristics that are computed using evolution analysis. We share KBQ-tool as open source in order to foster reproducibility of the experiments [3].

Figure A.1 shows the home page of the KBQ-tool. It contains four main modules: (1) Collect (2) Analyze (3) Visualize, and (4) Validate. Following we present detailed instructions for each module.

(1) Collect: This module performs data collection by selecting a class for any KB. Furthermore, data extraction is performed based on specific set of SPARQL queries for extracting summary statistics. This module collects periodic snapshots of a selected class by using a scheduler or by saving data manually. All the datasets are saved in CSV (comma-separated-value) file format. Table A.1 reports the header details of the CSV files.

[1] KBQ-Tool:http://datascience.ismb.it/shiny/KBQ/
[2] Demonstration: https://youtu.be/F02l7ImOZV8
[3] Source: https://github.com/KBQ/KBQ

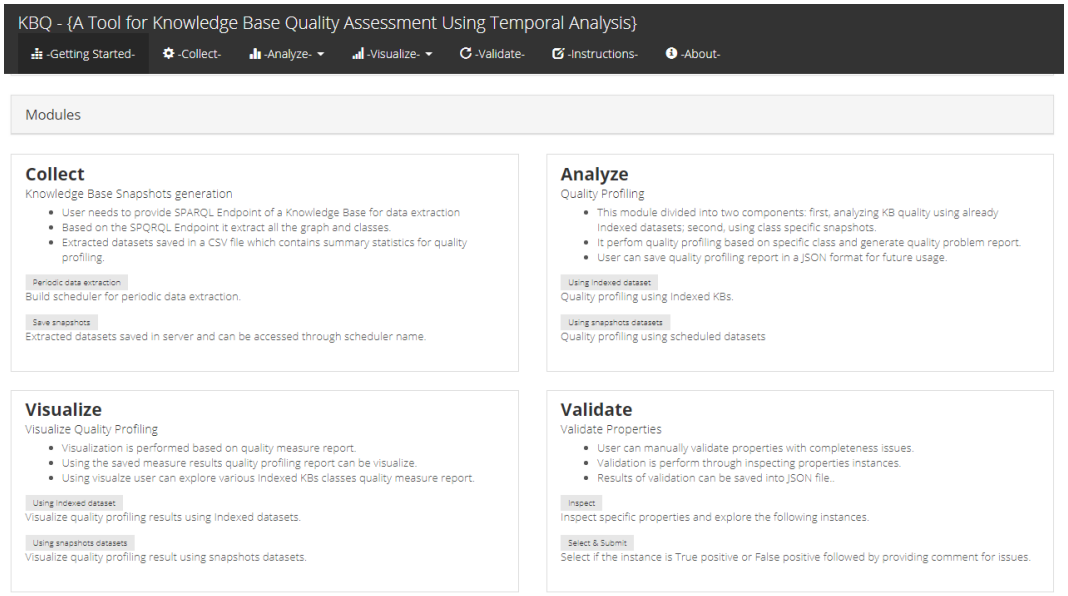


Fig. A.1 Home page of the KBQ-Tool.

Table A.1 Hedaer details of the CSV file.

Header	Description
Property	Name of a Property present in the Selected Class.
freq	Instance count of a property
Release	Date of the snapshot.
ClassName	Name of the selected class.
Graph	Name of the Graph.
Count	Entity Count of the Class.

For periodic data extraction, we have created a set of REST API. We use this API to create scheduler on the hosting server based on the selected class. Figure A.3 illustrates the scheduler architecture in the hosting server. REST APIs [4] are deployed in: http://datascience.ismb.it:9500/.

Figure A.2 reports the user interface for data collection module. Following we present instructions on how to build a scheduler in the data collection module.

[4]Source: https://github.com/rifat963/KBDataObservatory

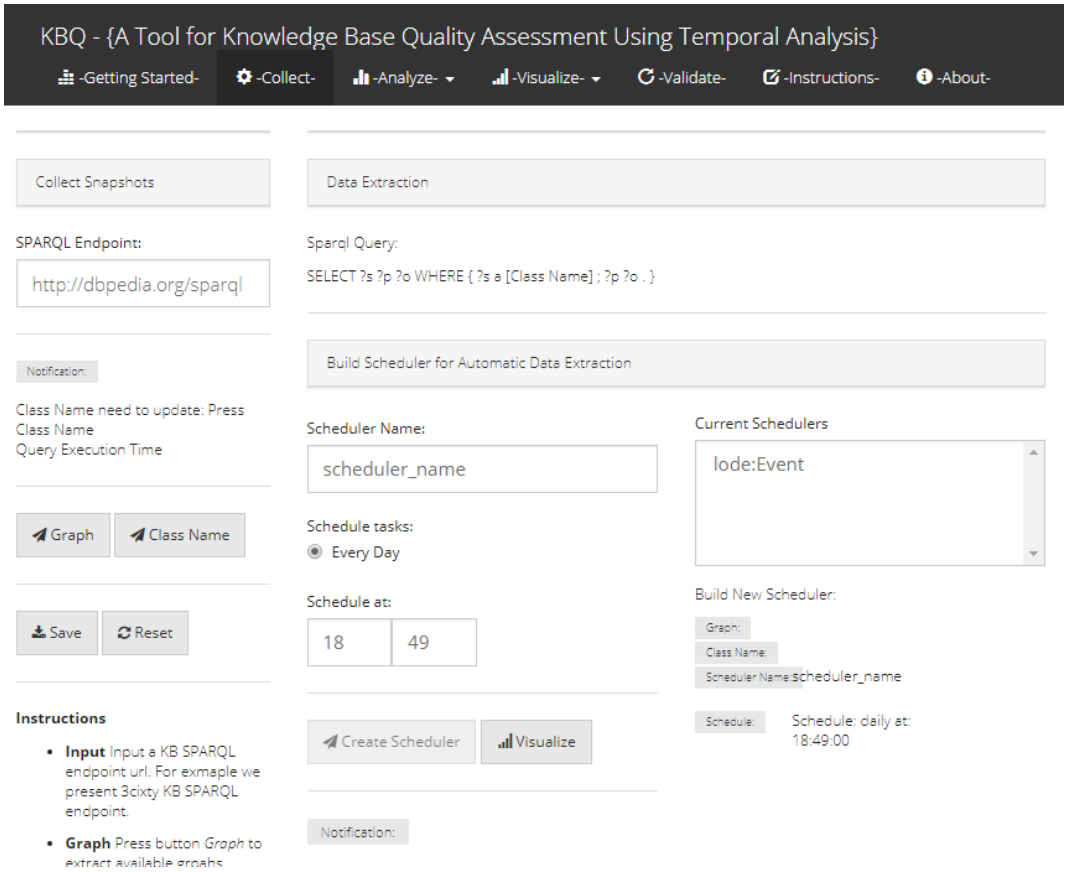


Fig. A.2 Example of inconsistent Wikipedia data.

(i) As an input, users need to provide a KB SPARQL endpoint URL. For exmaple, in the Figure A.2 we present DBpedia SPARQL Endpoint as an input.

(ii) User can collect available graphs present in the KB by selecting the button *"Graph"*.

(iii) For performing the data collection, a user needs to choose a class name. It is mandatory to select a class name for snapshots generation due to quality profiling is done based on an entity type. By selecting the button **"Class Name"** the user can extract all the available classes present in the KB.

(iv) In order to create schedulers, a user needs to provide a scheduler name and time for daily scheduling task. Then, he/she can press the button *"Create Scheduler"* to start the scheduler on the server. Any error present in the process is presented in the "Notification" labels. A simple example of data extraction using the REST API calls R code is presented in Listing A.1.

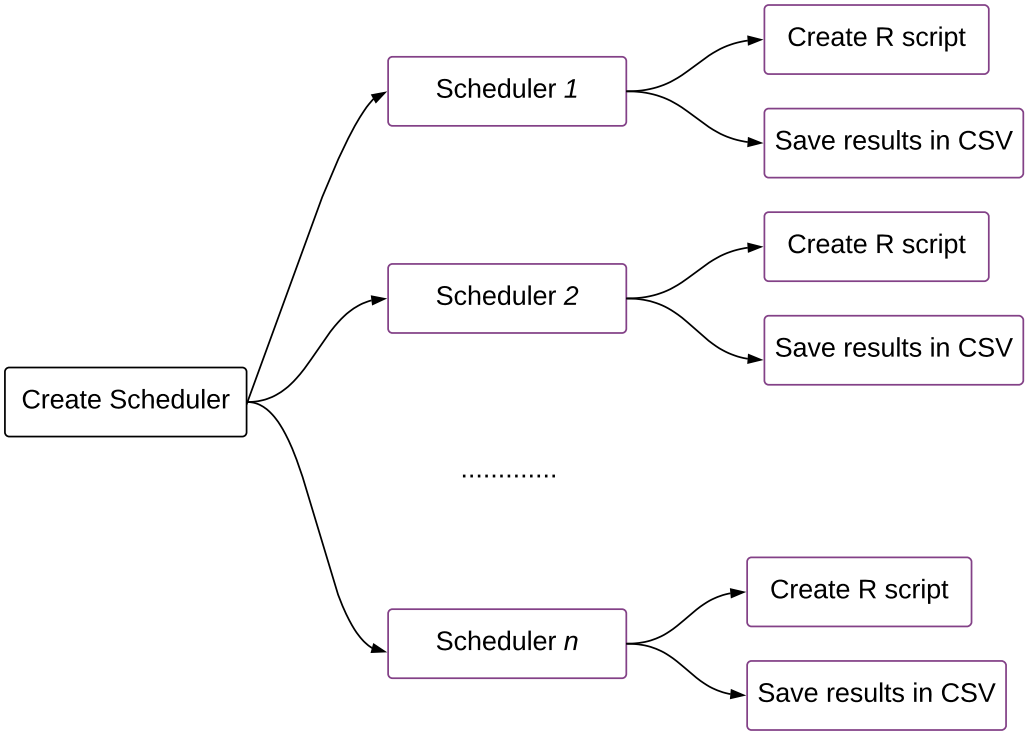


Fig. A.3 Scheduler Architecture.

Listing A.1 R code for scheduler task creation

```
#Input: Class Name, Graph, Sparql Endpoint.
#Process: Extract summary statistics using sparql.
#Return: Extracted data in JSON format

# DBpedia Sparql endpoint
endpoint<-"https://dbpedia.org/sparql"

schedulerName="scheduler_owl"
endpoint<-"https://dbpedia.org/sparql"

className<-"<http://www.w3.org/2002/07/owl#Class>"
className<-gsub("#", "%23", className)

graph<-"<http://dbpedia.org/resource/classes#>"
```

```
graph<-gsub("#", "%23", graph)


# It is necessary to create the R script before creating
   the Cron Job in the server

parm<-paste0("http://datascience.ismb.it:9500/","
   createRfile?filename=",schedulerName,"&className=",
                classname,"&endpoint=",endpoint,"&graph=",
                   graph)

responseCreateRfile<-GET(parm)
resCreateRfileContent<-content(responseCreateRfile)

# In the server duplicate scheduler will generate error.
   For this before createing scheduler please check
   current available schedulers.

parm<-"http://datascience.ismb.it:9500//readSchedulerIndex
   "
r<-tryCatch(GET(parm), error = function(e) return(NULL))
dt<-content(r)
DF<-fromJSON(dt[[1]])
DF$filename

#/createCornJob
#Input: Scheduler Name, time and frequency
#Process: Create scheduling task as a corn jon in the
   server based on the time and frequency.
#Return: Response as success or error

# Create the cron job
freq="daily"
time="14:25:00"
parm<-paste0("http://datascience.ismb.it:9500/","
   createCornJob?filename=",schedulerName,"&freq=",
            freq,"&time=",time)
r<-GET(parm)
```

```
content(r)
```

(2) Analyze: This module automatically performed the quality profiling and visualize the results using the indexed dataset (DBpedia) or by selecting the schedulers presented in the server. Quality profiling results are reported based on the four quality characteristics: Persistency, Historical Persistency, Consistency, and Completeness. Each of the quality characteristics present as a separate module to visualize the details of measurement results. For example, Figure A.4 illustrate the Historical Persistency result from analyze module. A user can save the quality profiling results in an HTML file. We visualize the analysis results into two parts:

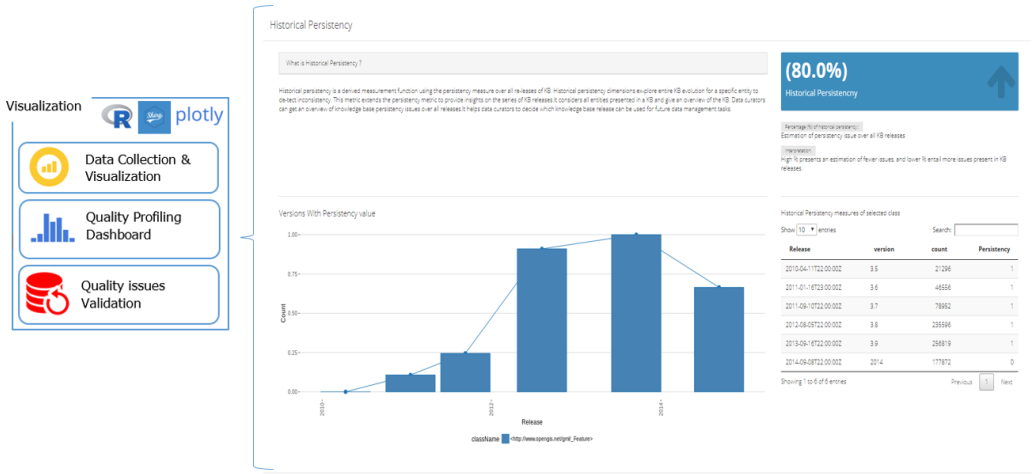


Fig. A.4 Example of Analyze module.

(i) Indexed KB: We considered DBpedia KB as an indexed KB and collected history of dataset updates using a dedicated SPARQL endpoint. To perform quality analysis, the user needs to select a class by using "Class Name" button. Then, he/she can perform quality profiling simple by selecting the button "Quality Profiling."

(ii) Sanapshots dataset: All the schedulers are presented in a list, and a user can run the quality profiling simply by selecting a scheduler.

(3) Visualize: We illustrated the results of quality profiling in two-stage: *(i)* summary of the quality profiling results is present together with a link to detail quality problem report, and *(ii)* data exploitation using schedeular and indexed KBs. A user can explore overall statistics on the saved classes simply by selecting a scheduler name presented in the list.

(4) Validate: This module is used for extracting, inspecting and commenting on the instances with quality issues. An end user can extract properties with quality issue after performing quality profiling. It follows the similar stages introduced in the manual evaluation phase of quality assessment process (Section 6.3.2). In the KBQ-tool, users can select a property then press the button "Inspect" to further explore the missing instances. Finally, user can save the validation report in a CSV file. Currently, this module can only performed validation for the Spanish version of DBpedia KB.

Appendix B

Publication List

The work presented in this thesis have been published in the following conference and journal conference proceedings, in which I have either been an author or a co-author.

Conference Proceedings

- Mohammad Rashid, Giuseppe Rizzo, Nandana Mihindukulasooriya, Marco Torchiano, and Oscar Corcho, "Knowledge Base Evolution Analysis: A Case Study in the Tourism Domain", In Proceedings of Workshops on Knowledge Graphs on Travel and Tourism co-located with 18th International Conference on Web Engineering (ICWE), Caceres,Spain, 2018 [95].
- Nandana Mihindukulasooriya, Mohammad Rashid, Giuseppe Rizzo, Raúl García-Castro, Oscar Corcho, and Marco Torchiano, "RDF Shape Induction using Knowledge Base Profiling", In Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC '18, pages 1952–1959, New York, NY, USA, 2018. ACM [26].
- Mohammad Rashid, Giuseppe Rizzo, Nandana Mihindukulasooriya, Marco Torchiano, and Oscar Corcho, "KBQ - A Tool for Knowledge Base Quality Assessment Using Temporal Analysis", In Proceedings of Workshops and Tutorials of the 9th International Conference on Knowledge Capture (K-CAP2017), Volume 2065 of CEUR Workshop Proceedings, Austin, Texas, 2017. CEUR-WS. org [24].

- Rashid, Mohammad, Torchiano Marco, "A systematic literature review of open data quality in practice", In Proceedings of 2nd Open Data Research Symposium (ODRS), Madrid, Spain, 2016 [96].

Journal article

- Mohammad Rashid, Giuseppe Rizzo, Marco Torchiano, Nandana Mihindukulasooriya, and Oscar Corcho, "A Quality Assessment Approach for Evolving Knowledge Bases." Accepted for publication in Special issue on Benchmarking Linked Data, Semantic Web Journal (2018) [2].

Journal article, under review

- Mohammad Rashid, Giuseppe Rizzo, Marco Torchiano, Nandana Mihindukulasooriya, and Oscar Corcho, "Completeness and Consistency Analysis for Evolving Knowledge Bases.", Journal of Web Semantics (2018) [97].

Other papers published during the PhD

- Rashid, Mohammad, Luca Ardito, Marco Torchiano, "Energy Consumption Analysis of Algorithms Implementations" In Proceedings of 9th International Symposium on Empirical Software Engineering and Measurement (ESEM), China, 2015 [98].

- Rashid, Mohammad, Luca Ardito, Marco Torchiano, "Energy Consumption Analysis of Image Encoding and Decoding Algorithms", In Proceedings of 4th International Workshop on Green and Sustainable Software (GREENS), 2015 [99].